PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 9/00	A1	(11) International Publication Number: WO 98/21648
		(43) International Publication Date: 22 May 1998 (22.05.98)

(21) International Application Number: PCT/US97/20627

(22) International Filing Date: 13 November 1997 (13.11.97)

(30) Priority Data:

08/749,926	13 November 1996 (13.11.96)	US
08/927,922	11 September 1997 (11.09.97)	US
08/964,751	5 November 1997 (05.11.97)	US

(71) Applicant *(for all designated States except US)*: PUMA TECHNOLOGY, INC. [US/US]; 2550 North First Street #500, San Jose, CA 95131 (US).

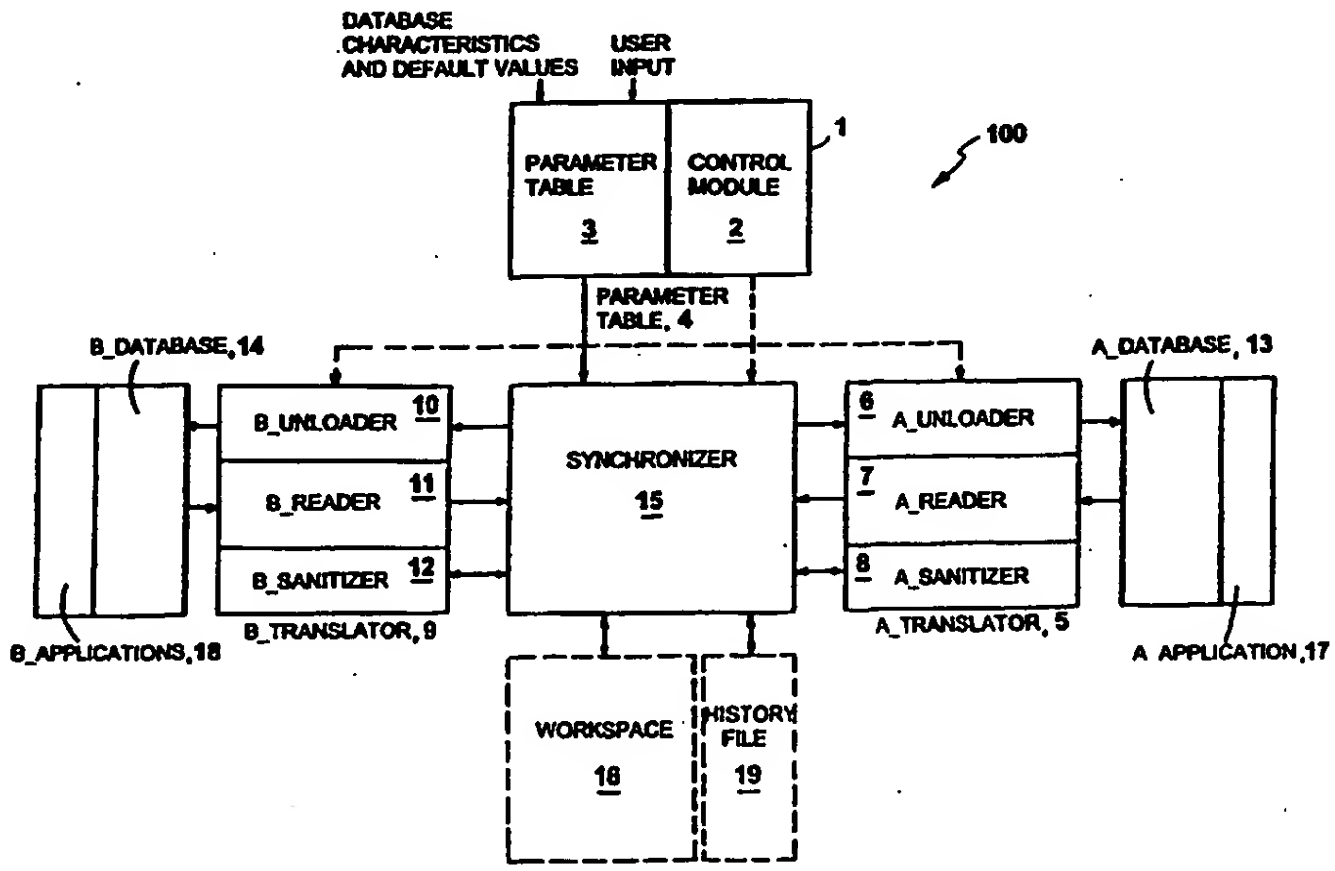
(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: BOOTHBY, David, J. [US/US]; 12 Thoreau Drive, Nashua, NH 03062 (US). DALEY, Robert, C. [US/US]; 13 Middle Dunstable Road, Nashua, NH 03062 (US). MARIEN, John, R. [US/US]; 23 Royal Crest Drive, No. 12, Nashua, NH 03062 (US).

(74) Agent: LEE, G., Roger; Fish & Richardson P.C., 225 Franklin Street, Boston, MA 02110–2804 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZW, ARIPO patent (GH, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) Title: SYNCHRONIZATION OF DATABASES



(57) Abstract

A computer implemented method and a computer program for synchronizing a first (13) and a second database (14), where data is provided for keeping track of whether the records of the first database have been added or changed since a previous synchronization. Based on the data reflecting whether the records of the first database have been added or changed since a previous synchronization, it is determined whether the records of the first database have been changed or added since the previous synchronization. The representative record is stored in a history file (19) which contains records reflecting the contents of records of the databases at the time of a previous synchronization.

DOCID: <WO___9821648A1_I_>

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

SYNCHRONIZATION OF DATABASES

Background

This invention relates to synchronizing incompatible databases.

Databases are collections of data entries which are organized, stored, and manipulated in a manner specified by applications known as database managers (hereinafter also referred to as "Applications"; hereinafter, the term "database" also refers to a database manager combined with a database proper). The manner in which database entries are organized in a database is known as the data structure of a database. There are generally two types of database managers. First are general purpose database managers in which the user determines (usually at the outset, but subject to future revisions) what the data structure is. These Applications often have their own programming language and provide great flexibility to the user. Second are special purpose database managers that are specifically designed to create and manage a database having a preset data structure. Examples of these special purpose database managers are various scheduling, diary, and contact manager Applications for desktop and handheld computers. Database managers organize the information in a database into records, with each record made up of fields. Fields and records of a database may have many different characteristics depending on the database manager's purpose and utility.

Databases can be said to be incompatible with one another when the data structure of one is not the same as the data structure of another, even though some of the content of the records is substantially the same. For example, one database may store names and addresses in the following fields: FIRST_NAME, LAST_NAME, and

ADDRESS. Another database may, however, store the same information with the following structure: NAME, STREET_NO., STREET_NAME, CITY_STATE, and ZIP. Although the content of the records is intended to contain the same kind of information, the organization of that information is completely different.

Often users of incompatible databases want to be able to synchronize them with one another. For example, in the context of scheduling and contact manager Applications, a person might use one Application on the desktop computer at work while another on his handheld computer or his laptop computer while away from work. It is desirable for many of these users to be able to synchronize the entries on one with entries on another. The U.S. patent and copending patent application of the assignee hereof, Puma Technology, Inc. of St. Jose, California (U.S. Patent No. 5,392,390 (hereinafter, "the '390 patent"); U.S. Application, Serial No. 08/371,194, filed on January 11, 1995, incorporated by reference herein) show two methods for synchronizing incompatible databases and solving some of the problems arising from incompatibility of databases.

Synchronization of two incompatible databases often requires comparison of their records so that they can be matched up prior to synchronization. This may require transferring records in one database from one computer to another. However, if the data transfer link between the two computers is slow, as for example is the case with current infrared ports, telephone modem, or small handheld computers, such a transfer increases the required time for synchronization by many folds.

Summary

In one general aspect, the invention provides for using certain design characteristics of certain

Applications to speed up the synchronization process. Some Applications provide information for keeping track of which records were changed, deleted, or added since the last synchronization. The invention uses these features to speed up the synchronization process by retrieving only those records which have been changed or added since a previous synchronization.

The invention provides a computer implemented method and a computer program for synchronizing a first and a second database. Based on data reflecting whether the records of the first database have been added or changed since a previous synchronization, it is determined whether the records of the first database have been changed or added since the previous synchronization. If one of the records of the first database has not been changed or added since the previous synchronization, a synchronization with records of the second database is performed using a record representative of the one record at the time of a previous synchronization. The representative record is stored in a history file which contains records reflecting the contents of records of the databases at the time of a previous synchronization.

Preferred embodiments of this aspect of the invention may include one or more of the following features. The data provided for keeping track of whether the records of the first database have been added or changed since a previous synchronization may be database generated data, stored in the records of the first database.

The computer generated data indicates the most recent date and time of when a record was created or changed. The computer generated data includes a flag set when a record is created or changed.

The first database provides further database generated data indicating which records were deleted

since the previous synchronization. Records of the history file corresponding to the deleted records are identified by performing a comparison of the further computer generated data with the history file. Records of the history file corresponding to records of the first database deleted since the previous synchronization are identified by performing a comparison of the records of the history file and a result of determining which of the records of the first database have been added or deleted.

The first database assigns a unique identification data to the records of the first database. A comparison of the records of the history file is made to the unique identification data of records in the first database that have been changed or added since the previous synchronization. The synchronization is completed using a result of the comparison.

Records of the first database and the history file are modified, added, or deleted based on the results of the synchronization. The history file contains records reflecting the contents of records of the databases at the time of the current synchronization.

Embodiments of this aspect of the invention may include one or more of the following advantages.

Some embodiments of the invention reduce the number of unchanged records that need to be read from the databases in order to synchronize the databases with one another. Typically a majority of the database records are unchanged. Therefore, not reading the unchanged record reduces the time required to synchronize the databases, especially where the data transfer link between the two computers which store the databases is slow.

In another general aspect, the invention features a computer implemented method for synchronizing a first

database located on a first computer and a second database located on a second computer. At the first computer, it is determined whether a record of the first database has been changed or added since a previous synchronization, using a first history file located on the first computer comprising records representative of records of the first database at the completion of the previous synchronization. If the record of the first database has not been changed or added since the previous synchronization, the first computer sends the second computer information which the second computer uses to identify the record of the first database to be unchanged.

The embodiments of this aspect of the invention may include one or more of the following features.

A second history file may be located on the second computer. The second history file contains records representative of records of the first database at the completion of the previous synchronization, where one of the representative records represents the record of the first database determined to be unchanged. Then, at the second computer, a synchronization of the second and first databases is performed using the one of the representative records.

The information sent from the first computer to the second computer can be used to locate the one of the representative records in the second history file. The second history file can store information in relation to the representative records and the one of the representative records in the second history file can be identified from that stored information. Additionally, the information sent from the first computer to the second computer can include information that matches the information stored in relation to the one of the representative records in the second history files.

The information sent to the second computer can include information identifying records other than the unchanged record. It can also include information identifying the changed record. It can also include information identifying the deleted records or added records. The information can also include a code based on at least a portion of the content of the record of the first database. The code may be a hash number. The information may be a code uniquely identifying the record of the first database. Such a code may be one assigned by the first database to the records.

In another aspect, the invention features a computer implemented method of identifying a record of a database. A record of the database is read. A code is assigned to the record of the database, the code being based on at least a portion of the content of the record of the first database. The code is then to identify the record at a later time.

The embodiments of this aspect of the invention may include one or more of the following features.

The code may be a hash number computed based on at least a portion of the content of a record of the first database.

The database is stored on a first computer and the code is transmitted to a second computer to identify the record to an application.

Advantages of these aspects of the invention may include one or more of the following advantages.

When synchronization is performed using the invention, a data transfer link, specially a slow data transfer link, is used efficiently, since unchanged records that are typically the majority of the records in a database are not transferred between the two computers. Hence, when synchronizing two databases on two different

computers, the time needed to synchronize the two databases is decreased

Also, when transmitting data from one computer to another, using a content based code, that requires less bandwidth for being transmitted and nonetheless identifies a record, results in a slow data transfer links being used more efficiently.

The invention may be implemented in hardware or software, or a combination of both. Preferably, the technique is implemented in computer programs executing on programmable computers that each include a processor, a storage medium readable by the processor (including volatile and non-volatile memory and/or storage elements), at least one input device, and at least one output device. Program code is applied to data entered using the input device to perform the functions described above and to generate output information. The output information is applied to one or more output devices.

Each program is preferably implemented in a high level procedural or object oriented programming language to communicate with a computer system. However, the programs can be implemented in assembly or machine language, if desired. In any case, the language may be a compiled or interpreted language.

Each such computer program is preferably stored on a storage medium or device (e.g., ROM or magnetic diskette) that is readable by a general or special purpose programmable computer for configuring and operating the computer when the storage medium or device is read by the computer to perform the procedures described in this document. The system may also be considered to be implemented as a computer-readable storage medium, configured with a computer program, where the storage medium so configured causes a computer to operate in a specific and predefined manner.

Other features and advantages of the invention will become apparent from the following description of preferred embodiments, including the drawings, and from the claims.

Brief Description of the Drawing

Figure 1 is a schematic drawing of the various modules constituting the preferred embodiment.

Figure 2 is a representation of the Workspace data array.

Figure 3 is the pseudocode for the Translation Engine Control Module.

Figure 4 is the pseudocode for loading records by a fast synchronization Translator.

Figure 5 is the pseudocode for loading records by a fast synchronization Translator.

Figure 6 is the pseudocode for loading records by a medium synchronization Translator and synchronizing by a synchronizer.

Figure 7 is the pseudocode for loading records by an alternative medium synchronization Translator.

Figure 8 is the pseudocode for an alternative method of loading records of a medium synchronization database.

Figure 9 shows two computers connected via data transfer link.

Figure 10 is a schematic drawing of the various modules constituting an alternative embodiment of the invention.

Figure 11 is pseudocode for the Translation Engine Control Module.

Figure 12 is pseudocode for a remote segment of a synchronization program when loading records from and unloading records to the remote database, when the database assigns unique IDs.

Figure 13 is pseudocode for a host segment of a synchronization program when loading records from and unloading records to the remote database, when the database assigns unique IDs.

Figure 14 is pseudocode for a remote segment of a synchronization program when loading records from and unloading records to the remote database, when the database does not assign unique IDs.

Figure 15 is pseudocode for a host segment of a synchronization program when loading records from and unloading records to the remote database, when the database assigns unique Ids.

Description

Briefly, a synchronization program 100 (Fig. 1) running on a computer loads records of two databases, e.g. a scheduling database on the computer and another one on a remote handheld or notebook computer. The synchronization program then synchronizes the records of those databases using a history file that contains records representative of the records of the two databases at the end of a previous synchronization. The synchronization program uses the history file to determine, for example, which records have been changed, added or deleted since the previous synchronization and which records of the two databases correspond to one another.

If one of the databases has the capability to provide database generated information or data which can be used to determine, for example, whether a record has been changed, added, or deleted since a previous synchronization, the synchronization program uses that information to determine whether a record has been changed, added, or deleted. Of course, that database generated information is less than the whole record of

the database. For example, that information may be a date and time stamp, or a flag, set when the record was last modified or when the record was added, whichever is later.

There are generally two types of such databases: "medium synchronization" and "fast synchronization" databases. A "fast synchronization" database is a database which provides information regarding changes, deletions, and additions to its records from one synchronization to the next. A fast synchronization database also assigns to each record of the database a unique identification code (i.e. a unique ID) which uniquely identifies that record. Unique IDs are required to accurately identify records over a period of time. A fast synchronization database also provides a mechanism for keeping track of which records are added, changed, or deleted from synchronization to synchronization, including a list of deleted records.

A "medium synchronization" database typically has more limited capabilities than a fast synchronization database for keeping track of addition, deletions, or changes. In short, a medium synchronization database does not keep track of deletions. Such a database however still has the capability to provide information regarding what records were added or modified since a previous synchronization. A medium synchronization database also provides unique IDs.

If the information provided by a database indicates that a record has not been changed or added since a previous synchronization, the synchronization program need not load that record and can use the history file to reconstruct the relevant contents of that record for synchronizing the two databases. The history file contains a copy of the relevant content of that record as the record was at the time of (e.g. at the end of) the

previous synchronization. Using the history file to reconstruct the record instead of loading the record can result in significant saving of time - where for example the data transfer link between the two computers is slow - since typically a majority of records in databases are unchanged records. The synchronization program thereby increases the efficiency of performing synchronization between two databases.

In order to better understand embodiments the invention described herein, we will briefly describe the overall structure of a synchronization program and the method it uses to synchronize databases, both of which are described in more detail in the following commonly owned U.S. patent applications, incorporated in their entirety by reference, "Synchronization of Recurring Records in Incompatible Databases", Serial no. 08/752,490, filed on November 13, 1996 (hereinafter, "application '490"); "Synchronization of Databases with Record Sanitizing and Intelligent Comparison," Serial no. 08/749,926, filed November 13, 1996 (hereinafter, "application '926"); "Synchronization of Databases with Date Range," Serial no. 08/748,645, filed November 13, 1996 (hereinafter, "application '645"). We will then describe specifically how the synchronization program operates when at least one of the databases provides information indicating that records have been changed, added or deleted since a previous synchronization.

Fig. 1 shows the relationship between the various modules of an embodiment of synchronization program 100. Translation Engine 1 comprises Control Module 2 and Parameters Table Generator 3. Control Module 2 is responsible for controlling the synchronizing process by instructing various modules to perform specific tasks on the records of the two databases being synchronized. (Fig. 3 shows the steps taken by this module.) The

Parameters Table Generator 3 is responsible for creating a Parameter_Table 4 which is used by all other modules for synchronizing the databases. Generally, Parameter_Table 4 stores various information which may be used by the modules of the synchronization program. The information stored in Parameter_Table 4 includes user preferences, the names and locations of the databases, and the names and locations of various files stored on disk including the name and location of the history file from the previous synchronization.

Synchronizer 15 has primary responsibility for carrying out the core synchronizing functions. It is a table-driven code which is capable of synchronizing various types of databases whose characteristics are provided in the Parameter_Table 4. Synchronizer 15 creates and uses the Workspace 16 (also shown in Fig. 2), which is a temporary data array used during the synchronization process.

A Translator 5 (A_Translator) is assigned to the A_database 13 and another Translator 9 (B_Translator) to the B_database 14. Each of the database Translators 5 and 9 comprises three modules: Reader modules 6 and 10 (A_Reader and B_Reader) which read the data from databases 13 and 14; Unloader modules 8 and 12 (A_Unloader and B_Unloader) which analyze and unload records from the Workspace into databases 13 and 14; and Sanitizing modules 7 and 11 (A_Sanitizer and B_Sanitizer) which analyze the records of the opposing database when they are loaded into the Workspace and modify them according to rules of data value of the modules's own database. Briefly stated, rules of data value are generally rules that define the permitted content of the fields of the records of a database. An example of such a rule would be that no more than 100 characters may be present in a field, or that content of a field

designating a priority for a "to do" item should be limited to 1, 2, or 3. Sanitizing a record is to change the content of the fields of a record of one database to conform to the rules of data value of another database. Rules of data value and sanitization are described in detail in the '490, '926 and '645 applications.

In the described embodiment, the modules of A_Translator 5 are designed specifically for interacting with A_database 13 and A_Application 17. Their design is specifically based on the record and field structures and the rules of data value imposed on them by the A_Application, the Application Program Interface (API) requirements and limitations of A_Application 17 and other characteristics of A_Database and A_Application. The same is true of the modules of B_Translator 9. These Translators are not able to interact with any other databases or Applications. They are only aware of the characteristics of the database and the Application for which they have been designed. Therefore, in the preferred embodiment, when the user chooses two Applications for synchronization, the Translation Engine chooses the two Translators which are able to interact with those Applications. In an alternate embodiment, the translators can be designed as table-driven codes, where a general Translator is able to interact with a variety of Applications and databases based on supplied parameters.

Fig. 3 is the pseudocode for the described embodiment of Control Module 2 of the Translation Engine 1. We will use this pseudocode to generally describe the steps taken when synchronizing two databases performed by the synchronization program 100. Control Module 2 first instructs the Parameter Table Generator 3 of Translation Engine 1 to create Parameter_Table 4 (Step 100). In step 102, the Translation Engine instructs synchronizer 15 to

load the history file. Synchronizer 15 in response creates the Workspace 16 data array and loads history file 19 into Workspace 16. History file 19 is a file that was saved at the end of last synchronization. It contains records representative of the records of the two databases at the end of the previous synchronization. The history file is necessary for use with the current synchronization. Generally, synchronization is a process of analyzing records from the A_Database and B_Database against the records of the history file to determine the changes, additions, and deletions in each of two databases since a previous synchronization and then determining what additions, deletions, or updates need be made to synchronize the records of the two databases. If no history file from a previous synchronization exists or the user chooses to synchronize not using the history file, step 102 is not performed.

Once the history file is loaded into the Workspace, Control Module 2 instructs B_Translator 13 to load the B_Database records (step 103). B_Reader module 11 of the B_Translator reads the B_database records and sends them to synchronizer 15 for writing into the Workspace. Following loading the B_Database records ("B_Records"), A_Sanitizer module 8 of A_Translator 5 sanitizes the B_Records in the Workspace (step 104). Control Module 2 of the Translation Engine 3 then instructs the A_Translator 5 to load the records from the A_Database (step 105). A_Reader module 7 of A_Translator 5 reads the A_Database records ("A_Records") and sends them to synchronizer 15 for writing into the Workspace. B_Sanitizer module 12 of B_Translator 9 then sanitizes the A_Records in the Workspace.

Records in the Workspace are stored according to the B_Database data structure. Therefore, as synchronizer 15 receives each A_record from the A_Reader

module 7 of the A_Translator 5, synchronizer 15 maps that record using an A→B_Map before writing the record into the next available spot in the Workspace. Since the A_records are mapped into the B_Record format, when the B_Sanitizer is instructed by Control Module 2 to begin sanitizing those records and starts asking for them from synchronizer 15, they already have the B_Database format. Therefore, synchronizer 15 does not need to map them before sending them to the B_Sanitizer module 12 of the B_Translator 19. For the same reason, there is no need for them to be mapped once they are sent back by the B_Sanitizer after having been sanitized. However, in the case of B_records, they must be mapped using a B→A map prior to being sent to A_sanitizer and then mapped back using an A→B map. At this point, all records are loaded into the Workspace.

Synchronizer 15 then performs a Conflict Analysis and Resolution ("CAAR") procedure on the records in the Workspace, which procedure is described in detail in the '490, '926 and '645 applications. Briefly, during this process, synchronizer 15 compares the records in the workspace and determines what synchronization actions should be taken. Synchronizer 15 processes the records, including comparing them to one another, in order to form them into groups of related records called corresponding item groups (CIGs). Each CIG may comprise at most one recurring or a group of related nonrecurring records from each of the databases and the history file. After forming CIGs from all records of the two databases, synchronizer 15 compares the records in each CIG with one another, determines their differences, and decides what synchronization action should be taken. Synchronization actions with respect to a record include updating, deleting, adding, or not modifying that record. For example, if after comparing the records in a CIG,

synchronizer 15 determines that the record from A_database is unchanged and the one from B_database is changed, synchronizer 15 determines that the A_database record should be changed to conform to the B_database record. Or, if both records are changed (an example of what we refer to as a "conflict" since there is no clear choice of synchronization action), synchronizer 15 may use a user-selected rule to decide what synchronization should be taken. The rule may require, for example, not modifying either of the records, changing the B_database record to conform to the A_database record, or asking the user to resolve conflicts.

When synchronizer 15 finishes performing CAAR on the records, synchronizer 15 would have determined what synchronization action should be taken with respect all records to be synchronized. The records may then be unloaded into their respective databases. The Translators will perform the specific synchronization actions to be taken with respect to the records of the databases. However, prior to doing so, the user is asked to confirm proceeding with unloading (steps 108-109). Up to this point, neither the databases nor the history file have been modified. The user may obtain through the Translation Engine's User Interface various information regarding what synchronization actions will be taken upon unloading.

If the user chooses to proceed with synchronization and to unload, the records are then unloaded. The Unloader modules 6,10 of the Translators 5,9 perform the unloading for the databases. Synchronizer 15 creates the history file and unloads the records into the history file. Control Module 2 of Translation Engine 1 first instructs the B_Translator to unload the records from Workspace into the B_Database. Following unloading of the B_RECORDS, Control Module 2

instructs the A_Translator to unload the A_Records from the Workspace (step 111). This unloading is done in the same way as it was done by the B_Translator. Control Module 2 next instructs synchronizer 15 to create a new history file (step 112). During unloading, the Unloader module of the A_Translator uses the A→B map the map the records in the Workspace into the format of A_database records.

At this point Synchronization is complete.

The speed of the above synchronization process can be improved in the case of databases which provide data for keeping track of which records have been changed, added, deleted, or left unchanged since a previous synchronization. As previously described, there are generally two types of such databases: "medium synchronization" and "fast synchronization" databases.

A "fast synchronization" database is a database which provides information regarding changes, deletions, and additions to its records from one synchronization to the next. Synchronization program 100 can use this information to speed up the synchronization process because records which have not been changed are not loaded from the database. Since the majority of records loaded by regular Translators are typically unchanged records, far fewer records are loaded from the database into Workspace than would otherwise be the case.

Certain features are required for a database to be a fast synchronization database. The database assigns each record of the database a unique ID and provides a mechanism for keeping track of which records are added, changed, or deleted from synchronization to synchronization, including a list of deleted records.

There are at least two ways to keep track of additions, changes, and deletions in a database.

First, some databases maintain one Dirty bit per record which is a boolean flag that is set when a record is created or modified and is cleared when a function for clearing Dirty bits is called. Some databases offer a Clear DirtyBits function that clears a Dirty bits of an individual record. Other databases offer a ClearDirtyBits function that clears the Dirty bits of all records in a database. The record-specific ClearDirtyBit function allows the described embodiment to use the database itself to keep track of additions and changes.

The global ClearDirtyBits function forces the described embodiment to clear all Dirty bits at the conclusion of every Synchronization. Then as database edits are made by the user in between synchronization, the affected records are marked as Dirty. When Synchronization is performed again, only the Dirty records are loaded.

Second, some databases maintain a date and time stamp of when the record was added or last time the record was modified. A Translator for such a database finds all records which were added or modified since the previous synchronization by searching for date and time stamps more recent than the Date&Time of the Last Synchronization.

A fast synchronization database must also keep track of deletions. This is done by maintaining a list of deleted records which can be read by a Translator.

A Translator sending fast synchronization database records to synchronizer 15 provides only records which have been changed, deleted, and added since the previous synchronization. Therefore, unlike a regular database Translator, a fast synchronization database Translator does not provide synchronizer 15 with unchanged records. Moreover, unlike a non-fast synchronization Translator, a

fast synchronization translator provides deleted records, which the regular Translators does not.

In order for such databases to be synchronized without resorting to treating them as regular databases (necessitating loading all records), synchronizer 15 transforms fast synchronization database records from the Translator into equivalent regular database records. These transformed records are then used by synchronizer 15 in the synchronization process. There are two transformations which are necessary. First, synchronizer 15 needs to transform deleted records received from the fast synchronization Translator into a regular database deletions. Second, synchronizer 15 needs to transform lack of output by the fast synchronization Translator into unchanged records.

Synchronization program 100 performs these transformations by using the history file. When the databases are synchronized for the first time, all records in the fast synchronization database are loaded into the history file. As changes, additions, and deletions are made to the fast synchronization database, during each of the subsequent synchronization, the same changes, additions, and deletions are made to the history file. Therefore, the history file at the end of each subsequent synchronization contains a copy of the data in the fast synchronization database.

When a fast synchronization database Translator supplies no input for a unique ID history file record, synchronizer 15 finds (i.e. identifies) the corresponding history file record in Workspace 16, copies it into Workspace 16, and treats the copied record as if it were loaded by the fast synchronization translator itself.

Referring to Fig. 4, steps 1050-1051, synchronizer 15 first verifies that there is an appropriate history file. Because synchronization of fast synchronization

databases relies heavily on the history file, it is important to ensure that the same history file as the last Synchronization is used. Moreover, the history file is the background against which the transformation of the Translator outputs into regular Translator outputs takes place. The history file keeps a date and time stamp of the previous synchronization. Each of the fast synchronization database (if able to) and the fast synchronization Translator also stores the same date and time stamp. The date and time stamp is used because it is unlikely that another history file will have exactly the same date and time entry, for the same two databases. The date and time stamp also identifies when last the fast synchronization database and the history file contained the same records.

At the start of synchronizing a fast synchronization database with another database, synchronizer 15 and the fast synchronization Translator compare date and time stamps. If the date and time stamp have changed since the previous synchronization, then the synchronization proceeds from scratch (step 1052). In a synchronization from scratch all records of the fast synchronization database are loaded into Workspace 16 and a history file is not used.

In the described embodiment, all records supplied as fast synchronization inputs have a special hidden field called _Delta, which carries a single-letter value - 'D' for deleted, 'A' for added, and 'C' for changed. Records are loaded by the fast synchronization Translator into Workspace 16 (step 1054). If necessary, the records are mapped when loaded. Records which are marked as changed or added are sanitized by the Translator for the other database, but deleted records are not because their field values are going to be deleted (step 1055). Orientation analysis, details of which are described in

the '490, '926, and '645 applications, is performed on the records. In general terms, during orientation analysis, deletions and changes to fast synchronization database records are joined with their history file counterparts in unique ID bearing CIGs (step 1107). These are CIGs in which at least one of the records is assigned a unique ID. Synchronizer 15 may treat these CIGs differently than other CIGs to improve the efficiency of the synchronization program, for example, by matching them to history file records based on their unique ID value instead of the content of record, as described in detail in applications '490, '926, and '645.

All history file records and their CIGs are now examined. If there is no corresponding record from the fast synchronization database, it means that the record was unchanged. A clone of the record is made, labelled as being from a fast synchronization database, and joined to the history file record's CIG. At this point the deleted fast synchronization database records marked as deleted are removed from CIGs (step 1109). The fast synchronization records marked as changed are joined in doubleton CIGs (i.e. CIGs entry records from two databases). Those marked as additions are singletons. At this point, the synchronization can proceed as if record of a unique ID bearing regular database were just loaded into Workspace 16.

Whenever records are loaded from a fast synchronization database, all records are loaded so that at the end of synchronization the history file will be the same as the fast synchronization database. Therefore, referring to Fig. 5, in order to perform date range limited synchronization, synchronizer 15 marks the records which fall outside the current and the previous date ranges. A record marked as an added, or during synchronizing from scratch, a record that falls outside

the current date range, it is marked as Out_Of_Range (steps 1101 and 1153-1154). This record will be written into the history file but not into the other database or take part in synchronization. When the fast synchronization database records are loaded from the history file, if they fall outside of the previous date range, they are marked as Bystander (steps 1152-1157). If a Bystander record forms a CIG with a fast synchronization record marked as a deletion or a change, the Bystander is marked with a Garbage flag because its field values serve no useful purpose any more: the record marked as DELETION should be deleted and the record marked as CHANGED should replace the Bystander history file record (step 1162).

History file records for which there are no inputs are transformed in the same manner as before (steps 1164-1165). If a Bystander record falls within the current date range, it is equivalent to a regular database record coming into the current date range. Therefore, the history file record is cloned and marked as a fast synchronization database record while the Bystander record is marked as Garbage (steps 1166-1171). Therefore, just like a new record of a regular database, the record has no history file record counterpart.

If the user selects to abort a synchronization or selects the option to ignore a conflict or conflicts in general, some of the records loaded from the fast synchronization database will not be accepted and recorded in the history file. Therefore, the Translator should provide that record again during the next synchronization. However, because fast synchronization Translators supply only records which have been changed, deleted, or added since the previous synchronization, the records which were not accepted will not be supplied since they remain unchanged. To circumvent this, the

fast synchronization Translator waits for an acknowledgement from synchronizer 15 that the record has been accepted.

In the case no such acknowledgement is received for a record, the Translator needs to be able to provide that record again to synchronizer 15. If the database allows resetting individual Dirty bits, the Translator merely does not set that bit. If not, the Translator keeps a separate file in which it keeps a record of which fast synchronization records were not accepted. The file may contain the unique IDs of those records. The Translator then uses that file to provide synchronizer 15 with those records during the next synchronization.

We will now describe synchronization in the case of medium synchronization databases. Medium synchronization databases have more limited capabilities than fast synchronization databases for keeping track of addition, deletions, or changes. Medium synchronization database do not keep track of deletions. They however still have the capability to provide information regarding what records were added or modified since a previous synchronization. This information typically takes the form of date and time stamps stored with the record or a Dirty bit, as previously described.

In the case of medium synchronization databases, the Translator provides synchronizer 15 with information indicating which records have been added/changed and which records are unchanged. Based on this information and the history file, synchronizer 15 determines which records were deleted. Synchronizer 15 assumes that those records in the history file for which no matching unique ID was obtained from the database are deleted records and flags them as such. In identifying a record as having been deleted from the database, synchronizer 15 can easily determine what synchronization action should be

taken with respect to the counterpart of that record in the other database; such action may be to delete the other record. Those records whose unique IDs do not match the unique IDs of the history file records are new records and synchronizer 15 marks them specifically as "added" records.

Synchronizing medium synchronization databases, like in the case of fast synchronization databases, relies on the history file. Therefore, the description of the history file and its treatment in the case of fast synchronization databases equally applies to the case of medium synchronization databases.

Fig. 6 shows the pseudocode for the steps taken by a translator for a medium synchronization database that uses a date and time stamp to provide information regarding whether a record has been changed or added since a previous synchronization. For every record in the database, the translator gets the unique ID and the date and time stamp of that record (steps 1200-1202). If the date and time stamp is prior to or at the same time as the date and time of the previous synchronization, translator determines the records to be unchanged (steps 1203-1204). As described in the case of fast synchronization database, each record in the workspace has a hidden field called _Delta, which carries a single-letter value - 'D' for deleted, 'A' for added, and 'C' for changed. In the case of medium synchronization, the hidden field can contain four values, 'D' for deleted, 'A' for added, 'C' for changed, and 'U' for unchanged. Therefore, once the translator determines that the record is unchanged, the translator stores an appropriate value to indicate to the synchronizer that the record is unchanged (step 1205). Records that are unchanged need not be loaded. A copy of the relevant data in the record is contained in the history file. Synchronizer 15

replicates the data of the unchanged record using the content of the history file, in the same manner as in the case of fast synchronization databases. By relevant, we mean data in the fields that are synchronized.

If a record is not unchanged, then the record has either been modified and added since the previous synchronization. The translator sets value of the hidden field as 'C' which indicates to synchronizer 15 that the record was either added or changed (step 1206). The translator then loads the changed or added record, from the database, field by field (steps 1207-1209). The translator then loads the record into the workspace (step 1211).

After the translator has loaded all the records in the database, there are in essence two logical lists in workspace 16. One is the list of records identified as changed/added since a previous synchronization. Second is the list of records identified as unchanged since a previous synchronization. Synchronizer 15 compares the unique IDs of the records in these two lists with the unique IDs in the history file (step 1213). (In the case of date range limited synchronization, the whole database would be loaded but only those records which are in the date range will be synchronized, as was the case for fast synchronization databases.) Based on the comparison, synchronizer 15 determines which records in the history file are no longer present in the database. Synchronizer 15 determines that these records have been deleted and sets the hidden field value for these records as 'D' (step 1214). Synchronizer 15 processes these records in the same manner as when synchronizer 15 receives an indication from a fast synchronization database that a records has been deleted. The synchronizer also compares the unique IDs changed records to the unique IDs of the records of the history file (step 1215). The

Synchronizer determines that those unique IDs which are not present in the history file belong to newly added records (step 1216). In that case, synchronizer 15 changes the value stored in the hidden filed value to 'A' to indicate that the record was newly added.

At this point the output of the medium synchronization database has been essentially transformed into output of a fast synchronization database. Synchronizer 15 then proceeds to synchronize the records in the same manner as synchronizer 15 does for fast synchronization databases, since the synchronizer has information as to which records have been changed/added, deleted, and unchanged (step 1217).

At the end of synchronization, during unloading, the history file is updated in the same manner as is the case for fast synchronization database, to ensure the history file records reflect the content of all records of the database at the end of synchronization (step 1218). During unloading, as in the case of fast synchronization, the translators wait for acknowledgement from the databases during unloading.

Other embodiments are within the following claims.

For example, Fig. 7 shows the pseudocode for another embodiment in which the medium synchronization database is able to perform answer queries based on specified criterion. An example of such a criterion would be "changed records after 1/1/1997" in response to which the database provides a list of records matching that criterion. The difference between this embodiment and the embodiment described in reference to Fig. 6 is as follows. The Translator, in step 1250, queries records that have date and time stamps that are prior to or at the same time as the date and time of the last synchronization. The Translator determines these records to be unchanged. The Translator also queries the

database for records that have date and time stamps that are subsequent to the date and time stamp of the last synchronization. Translator determines these records to be changed or added since the last synchronization. The rest of the processing by the Translator and synchronizer are the same as the embodiment described in reference to Fig. 6.

In the case of those databases that provide Dirty bits instead of date and time stamps, the Translator determines which records have their Dirty bits set. Those that have their Dirty bit set are changed or added records while those that do not have their Dirty bits set are unchanged records. The rest of the processing by the Translator and synchronizer are the same as the embodiment described in reference to Fig. 6.

Fig. 8 shows another embodiment for synchronizing a medium synchronization database. In this case, for each record (step 1350), the translator loads the unique ID and the date and time stamp or Dirty bit, as may be the case. If the record has not been modified or added since the previous synchronization (step 1353), the translator calls a special function (PutUnchangedRecord) of synchronizer 15 and also supplies the unique ID of the unchanged record to the synchronizer (step 1354). In response to the function call, synchronizer 15 searches the history file for the supplied unique ID (step 1355) and clones the matching history file record (step 1356).

If the record has been modified or added since the previous synchronization (step 1357), the record is loaded from the database (step 1358-1360) and loaded into the workspace (step 1361). Synchronization then proceeds as if the records of a regular database (i.e. a database that does not provide information that may be used to keep track of whether records of the database have been changed, added or deleted) are loaded, as described

briefly above and in more detail in '490, '926, and '645 applications. The deleted records, in this embodiment, are then determined as the two databases are synchronized.

In the above described embodiments, a database provides database generated data that may be used to keep track of status of the record (e.g. changed, added, modified, deleted). However, it is also possible to provide a program, for example running on a remote computer, that keeps track of the status of the records of a database. That remote program may then provide the synchronization program only those records that have been changed or added and use data less than the full record to identify those records that are unchanged or have been deleted. We will briefly describe an embodiment of such a program, which is described in detail in the commonly assigned copending U.S. patent application, incorporated herein in its entirety by reference, entitled "DISTRIBUTED SYNCHRONIZATION OF DATABASES", filed on September 11, 1997, serial no. 08/927,922.

Briefly, referring to Figs. 9 and 10, a synchronization program , according to the embodiments described here, has a host segment 28 and a remote segment 26 which run on a host computer 20 and a remote computer 22, respectively. The two computer are connected together via a data transfer link 24 enabling them to transfer data between them. Data transfer link 24 may be a slow data transfer link such as a serial infrared links, serial cables, modems and telephone lines, or other such data transfer links. A host database 13 and a remote database 14, e.g. scheduling databases, are stored on remote computer 22 and host computer 20, respectively.

Generally, in some instances, both computers on which the two databases run are capable of running

programs other than a database, as in the case of, for example, general purpose computers such as desktop and notebook computers, or handheld computers having sufficient memory and processing power. In such a case, the synchronization program may be distributed between the two computers so as to, for example, increase the efficiency of using of a slow data transfer link between the two machines.

Briefly, at remote computer 22, remote segment 26 of the synchronization program loads records of remote database 13. Remote segment 26 then determines which records of the remote database have been changed/added, deleted or left unchanged since a previous synchronization. If the remote database assigns unique identification codes (i.e. unique ID) to its records, remote segment 26 can further differentiate between records than have been added and those than have been changed since the previous synchronization. Remote segment 26 uses a remote history file 30 which stores data representing or reflecting the records of the database at the completion of the previous synchronization. This data may be a copy of remote database 13. It may also be hash numbers for each of the records of the remote database. If the remote database assigns unique IDs, the remote history file may contain those unique IDs together with the hash numbers of the records corresponding to the stored unique IDs.

Remote segment 26 sends those records of the remote database that have been changed or added to the host segment or the host computer. However, the remote segment does not send the unchanged or deleted records to the host computer. Instead, the remote segment sends a flag indicating the status of the record (e.g. unchanged or changed) and some data or information that uniquely identifies the record to the host segment. This data or

information may be a hash number of all or selected fields in the record at the completion of the last synchronization. It may also be the unique ID assigned to the record by the remote database, if the database assigns one to its records.

Host segment 28 uses the received information or data that uniquely identifies the unchanged record to access a record in host history file 19 that corresponds to the received information or data. This record contains a copy of the data of the remote database record that the remote segment found to have been unchanged. Host segment 19 then uses this record to synchronize the databases by comparing it to the records of host database 14. After synchronization, the remote and host history files and the databases are updated. Since the unchanged records which typically constitute most of the records of a database are not transferred to the host computer, a data transfer link, specially a slow data transfer link, is used with increased efficiency.

We will describe two embodiments of a distributed synchronization program. We will first describe in general terms the overall structure of the distributed synchronization program in reference to Figs. 10 and 2 which is common to both embodiments. We will then describe then the first and second embodiments performing a distributed synchronization in reference to Fig. 11-15.

Fig. 10 shows the relationship between the various modules of an embodiment of a distributed synchronization program. Translation Engine 1 comprises a Control Module 2 that is responsible for controlling the synchronizing process by instructing various modules to perform specific tasks on the records of the two databases being synchronized. The Control Module 2 also provides data that affects the specific operation of the various

components of the synchronization program, such as the name of the databases being synchronized and user preferences. Fig. 11 is the pseudocode of the steps taken by this module. The Synchronizer 15 has primary responsibility for carrying out the core synchronizing functions. It is a table-driven code which is capable of synchronizing various types of databases whose characteristics are provided by control module 2. The Synchronizer creates and uses a host workspace 16 (shown in detail in Fig. 2), which is a temporary data array used during the synchronization process.

A host translator 9 includes two modules: a reader module 10 which reads the data from the host database 14 and an unloader module 10 which analyzes and unloads records from the host workspace into the host database 14. Remote segment 26 also has similar modules for reading and unloading data from the remote database. The remote segment is designed specifically for interacting with remote database 13. The design of the remote segment is specifically based on the record and field structure of the remote database and remote database's Application Program Interface (API) requirements and limitations and other characteristics of the remote database. Similarly host translator 9 is designed specifically for the host database. The remote segment and host translator are not able to interact with any other databases or Applications. They are only aware of the characteristics of the databases for which they have been designed. In an alternate embodiment, the host translator and the remote segment can be designed as a table-driven code, where a general Translator is able to interact with a variety of databases based on the parameters supplied by, for example, the Control Module 2. It should be noted that the remote segment and host

translator may be designed in various ways and still perform the tasks set out in this embodiment.

Fig. 11 is the pseudocode for the operation of Control Module 2 of the Translation Engine 1. We will use this pseudocode to generally describe distributed synchronization according to the invention. Control Module 2 first initializes itself and specifies the current user options to various modules (Step 401). In step 402, control module 2 instructs the Synchronizer to load host history file 19. Synchronizer 15 in response creates host workspace 16 data array and loads host history file 19 into host workspace 16. Host history file 19 is a file that was saved at the end of last synchronization and contains records representative of the records of the two databases at the end of the previous synchronization. Typically, the host history file contains a copy of the results of the previous synchronization of the synchronized records of the two databases. It should be noted that the content of the records of the history file may be limited only to those fields that are synchronized and the data may be translated and stored in a format different than that of the remote database or the host database. This data can be used to reconstruct the content of the records of the remote database as they were at the end of the previous synchronization. The host history file is generally used to determine changes to the databases since a previous synchronization and also to recreate records not sent from the remote segment, as will be described in detail below. If no history file from a previous synchronization exists or the user chooses to synchronize without using the history file, in step 402 the synchronizer does not load a history file. In that case, all the records from both databases will be loaded into the host workspace. We will describe the rest of the

operation of the control module as if a history file exists and will be used.

Once the History File is loaded into the host workspace, Control Module 2 instructs host translator 13 to load the host database records (step 403). Host Reader module 11 of the host Translator reads the host database records and sends them to the Synchronizer for writing into the host workspace.

Control Module 2 then instructs remote segment to send the records of the remote database (step 404). Remote segment 26 reads the remote database records and sends them to Synchronizer 15 for writing into the host workspace. The actions taken by the synchronizer and the remote segment in response to step 404 will be described in detail in reference to Figs. 12-15, below.

Records in the host workspace are stored according to either the host database or the remote database data structures. Therefore, as synchronizer 15 receives each record, the Synchronizer maps that record using the appropriate record map (i.e. either a remote database to host database record map or a host database to remote database record map) before writing the record into the next available spot in the host workspace. Mapping may be performed by other modules, e.g. the remote segment. The records may also be "translated", i.e. cast into a format which synchronizer can use (a "translation" method is described in the '390 patent). For example, a date stored as "April 1, 97" may be translated into a format preferred by the synchronizer, e.g. "4-1-97".

Control module 2 then instructs the Synchronizer to perform a Conflict Analysis and Resolution ("CAAR") procedure on the records in the host workspace (step 405), which procedure is described in detail in the following applications of the assignee hereof, Puma Technology, Inc. of St. Jose, California, incorporated by

reference in their entirety including any appendices: "Synchronization of Recurring Records in Incompatible Databases", Serial no. 08/752,490, filed on November 13, 1996 (hereinafter, "'490 application"); "Synchronization of Databases with Record Sanitizing and Intelligent Comparison," Serial no. 08/749,926, filed November 13, 1996 (hereinafter, "'926 application"); "Synchronization of Databases with Date Range," Serial no. 08/748,645, filed November 13, 1996 (hereinafter, "'645 application"). Generally, synchronization is a process of analyzing records from the remote database and host database against the records of the history file to determine the changes, additions, and deletions in each of the two databases since the previous synchronization and what additions, deletions, or updates need be made to the databases to synchronize the records of the databases. Briefly, during CAAR, the synchronization engine (i.e. the Synchronizer) compares the records in the host workspace and determines what synchronizing actions should be taken. The synchronization engine processes the records, including comparing them to one another, in order to form them into groups of related records. Each of these groups may comprise at most one recurring or a group of related nonrecurring records from each of the databases and history file. After forming these groups from all records of the two databases, the Synchronizer determines what synchronization action should be taken. To do this, the Synchronizer compares them, determines their differences, and decides what synchronization action is appropriate or asks the user what action should be taken. The synchronizer then associates with that record, the specific "action" (e.g. add, update or delete) that must be taken with respect to that record in that record's database. During "CAAR", the user may select not to synchronize a particular

record with the other database. We will describe below in detail the steps performed by the synchronizer and the remote segment in response to the output of CAAR as the output relates to the remote database.

Once Synchronizer 15 finishes performing CAAR on the records, the records may be unloaded or written into their respective databases, including any additions, updates, or deletions. However, prior to doing so, the user is asked to confirm proceeding with unloading (steps 108-109). Up to this point, neither the databases nor the History File have been modified. The user may obtain through the Control Module's Graphical User Interface (GUI) various information regarding what will transpire upon unloading.

If the user chooses to proceed with synchronization and to unload, the records are then unloaded in order into the host database, the remote database and the History File. The Synchronizer in conjunction with the host translator and the remote segment perform the unloading for the databases. Synchronizer 15 creates a host history File and unloads the records into it. Control Module 2 first instructs the host translator to unload the records from host workspace into the host database. Following unloading of the host records, Control Module 2 instructs the synchronizer and the remote segment to unload the remote records from the host workspace (step 409). We will describe in detail below, in reference to Figs. 12-15, the specific actions taken by Synchronizer 15 and remote segment 26 in order to unload data from the host workspace into the remote database and the update remote history file 28. Control Module 2 next instructs the Synchronizer to create a new History File (step 112). At this point Synchronization is complete.

Referring to Figs. 12-15, we will now describe the actions taken by the remote segment in coordination with the Synchronizer in response to the instructions from control module 2 in step 404 to load records of the remote database and in step 409 to unload the records of the remote database from the host workspace. Specifically, we will describe two embodiments. In the case of the first embodiment, the remote database assigns unique identification codes (i.e. unique IDs) to each of its records as they are created. In the case of the second embodiment, the remote database does not assign unique IDs to its records. Fig. 12 is the pseudocode for the steps taken by the remote segment while Fig. 13 is the pseudocode for the steps taken by the Synchronizer in the case of the second embodiment. Similarly, Fig. 14 is the pseudocode for the steps taken by the remote segment while Fig. 15 is the pseudocode for the steps taken by the Synchronizer in the case of the first embodiment.

Briefly, the remote segment determines which records have been changed/added, deleted or left unchanged since a previous synchronization. The remote segment uses a history file located on the remote computer ("remote history file") to determine which records may have been changed/added, deleted or left unchanged since a previous synchronization. The remote segment essentially can translate outputs of any database into outputs of a fast synchronization database which is a type of database that generally supplies information as to which of its records have been changed, added, deleted, or left unchanged. Fast synchronization databases and an example of a method of synchronizing them with other databases is described in detail in the '490, '926 & '645 applications. Therefore, for example, this method of distributed synchronization may also be

implemented with any synchronization program that is able to synchronize such databases.

Generally, the remote segment sends the host segment, over the data transfer link, only the content of those records that have been changed or newly added. As for unchanged records, the history file contains all necessary information to recreate or synchronize those records, if needed. Therefore, it is not necessary to transfer those records to the host segment. Only some data or identification code that uniquely identifies the record to the Synchronizer need be transferred for such a record. Since the majority of records are typically unchanged records, not transferring them over the slow data transfer link improves the efficiency of the synchronization process.

After all necessary information has been transferred to the host segment, the Synchronizer synchronizes the databases. Following synchronization, the host segment transfers information necessary to update the remote database and the remote history file to the remote segment. The remote segment then updates its history file and the remote database.

Since both the host and remote segments rely heavily on history files to enable distributed synchronization, it is important that the host and remote segments use history files that correspond to one another, i.e. both contain records corresponding to a previous synchronization of the same two databases. In the described embodiment, the remote and host history files are named using a common naming convention. The name of a file is made up of six components:

1) Name or ID of the host computer, which may be an assigned name such as an assigned GUID in the case of operating systems by Microsoft Corporation of Redmond,

Washington, or UUID in the case of operating systems by Open Software Foundation;

2) Name or ID of the host database application, e.g. trademark designations "Lotus Organizer" or "Microsoft Schedule+";

3) Name or ID of the host database file as stored on the long term storage (e.g. hard disk drive) of the host computer, e.g. "My Calendar";

4) Name or ID of the remote computer;

5) Name or ID of the remote database application; and

6) Name or ID of the remote database.

Therefore, the remote segment and the host segment ensure that the host history file have the same name. Moreover, each of the history files have the date and time stamp of the previous synchronization. The remote segment and synchronizer use this to ensure that the history files from the same previous synchronization of the two databases are used.

Having described in general terms the actions taken by the remote segment in coordination with the Synchronizer in response to the instructions from control module 2 in steps 404 and 409 (Fig. 11), we will now describe in detail a first embodiment of their operation for the case where the remote database assigns unique IDs to its records. We will do so in reference to Figs. 12 and 13.

Fig. 12 is the pseudocode for steps taken by the remote segment in response to the instruction by control module in step 404 to load the remote database records into the host workspace (Fig. 11). The remote segment first initializes (i.e. creates) a remote workspace in the remote computer (step 501). The remote segment then compares the name of the host history file with the name of any remote history file in the remote computer. If

the remote segment finds a remote history file that matches the host history file (i.e. a remote history file that matches the host history file) (step 502), then the remote segment examine the date and time stamp of the host and remote history files. If the date and time stamp in the remote history file matches the one in the host history file (step 503), then the remote segment determines that two history files correspond to one another. Hence, the remote segment loads the remote history file into the remote workspace.

In general, if matching history files do not exist on the remote and host computers, the remote segment transfers all remote database records to the host computer. Therefore, if the name of the host and remote history files match but the date and time stamps do not match (step 505), then the remote segment assumes that remote history file is not the correct remote history file to be used. The remote segment removes that history file (step 506) and transfers all remote database records to the host computer (step 507). If no remote history file matches the host history file (step 508), then the remote segment assumes an appropriate remote history file does not exist. The remote segment transfers all the records to the host computer (step 509). To transfer all the records in the above steps, the remote segment first loads and stores all records of the remote database in the remote workspace. The remote segment then transfers all records in the remote database to the host segment. If remote segment transfers all the records of the remote database to the host segment in either step 504 or 509, then the remote will go to step 528. It should be noted that the host segment will use the host history file, if one exists, to perform the synchronization.

If an appropriate remote history file exists - i.e. conditions of steps 501 and 504 are satisfied - the

remote history file is loaded into the work space. It is then used to "filter" out information that need not be sent to the host segment since it already exists on the host segment. Generally, the history files on the remote and history files are used to store information representative of the remote database at the end of the previous synchronization. The records of the remote history file in the first embodiment contain the unique ID of the records and hash numbers of those records at the completion of a prior synchronization. In other embodiments, the remote history file may contain some or all of the field values of the records of the remote database.

Hashing may be described as converting any data, such as a string of characters, into a more compacted format, such as a number, meant to represent that string of characters. It may be considered to be a content-based encoding technique. The hashed values may be used as a surrogate for a hashed string of characters, for example, to compare strings. An example of a hashing algorithm is to calculate the following sum for every characters in a character string:

sum = character + (31 * sum),

where character is the number stored in the memory to represent that character (e.g. an Ascii value). (It should be noted that there are many ways of hashing data.) At the end of the computation, sum contains the hash number for that string of characters. In the described embodiments, the hash number is a 32 bit number and therefore can have a value between 2^{32} different values. Because the expected number of records is much less than this number, the probability of two different records having the same hash value is small. Therefore, hash numbers can be used to perform comparisons instead of comparing the non-hashed data or a preliminary check

before comparing the data, with relatively low risk inaccurate comparison. We have also use hash numbers as a unique identification code, which will be described in the second embodiment.

The remote segment uses the remote history file to determine whether a record has been changed, deleted, or added since a previous synchronization. Therefore, for records that are unchanged, which typically constitute the majority of records in a database, the remote segment sends information that the host segment can use to identify the matching records in the host history file. That matching history file record contains the same data as necessary to use for synchronization as that on the remote database since the record is unchanged. Therefore, there is no need to send the whole record. In essence, the remote segment uses the remote history file to filter out information that is already contained in the host history file and sending only those records that have been changed or added. In some embodiments, the remote history file may contain all the field values of the records of the remote database. In those embodiments, the remote segment can determine not only which records have been changed but more specifically which field values have been changed. In that case, the remote segment can determine and then send only those field values that have been changed, further increasing the efficiency of using the slow data transfer link.

We will now describe this process in detail. In the described embodiment, for each record of the remote database (step 515), the remote segment loads the field values, including the unique ID, of the record into the remote workspace (step 512). As the records are loaded, they are translated (e.g. "translated" as described in the '390 patent) into a universal format for the remote workspace. The records will be translated back into the

format of the remote database as they are written into the remote database. The remote segment also computes a hash number based on all or selected (e.g. the fields to be synchronized) field values (step 513). In the described embodiment, the hashing number is a 32 bit number. The fields on which the hash number is based on remain the same for all synchronizations relying on this remote history file. The host segment also performs a hash on the same fields. If the fields which are hashed changes, the hash number of unchanged records would not remain the same from one synchronization to the next.

If the unique ID matches one of the unique IDs of records in the remote history file (step 515), then the record was present during the previous synchronization. That record could either be a changed record or an unchanged record. If the computed hash number for the record matches the hash number of the record in the history file (step 516), then the remote segment assumes that the record has not been changed since the previous synchronization and therefore can be created by the host segment from the host history file. The remote segment will take no action (step 517). In other embodiments, the remote segment can send the unique ID and a flag indicating that the record is unchanged to the host segment.

If the computed hash number does not match that of the history file record (step 518), the remote segment assumes that the record has been changed since a previous synchronization. Therefore, the remote segment sends the host computer the field values including the unique ID and a "changed" flag (step 519). In some embodiments, only those field values that have been changed since the previous synchronization will be sent, as described above. The remote segment then creates a new entry for the changed record in the history file (step 520) and

marks the record as unacknowledged (step 521), the purpose and function of which we will now briefly describe and is also described in the '490, '926 and '645 applications.

Generally, the remote segment does not change an entry in the remote history file, until it receives an instruction indicating that the host segment has synchronized and updated the host database with that record. This is done so that if for any reason (e.g. user does not want to update that record of the host database as described above) the host database is not synchronized with that record, the remote segment will not treat that record as unchanged during the next synchronization. The acknowledgement may take the form of an "acknowledgment" flag or an "action" instruction which instructs the remote segment to add, update, or delete that record of the remote database, as described above. Therefore, for each changed and deleted record, the remote segment creates a new entry and marks the entry as "unacknowledged". If an "acknowledgment" flag is received, the old history file record is deleted. If an "acknowledgement" flag is not received, the new workspace entry is deleted. The steps will be described further below.

If in step 515 the remote segment determines that the unique ID of the loaded record does not match any of the unique IDs stored in the records of the history file (step 521), the remote segment assumes that the record loaded from the remote database has been newly added. Therefore, the remote segment sends the host segment a copy of the field values of those fields of the record to be synchronized (which may be all or less than all the fields) together with an "added" flag (step 524). As in the case of a changed record, the remote segment creates a new remote workspace entry and enters the unique ID and

hash value of the record (step 525). The new entry is marked as unacknowledged (step 526).

After all the records have been loaded (step 528), the remote database determines that unique IDs in the history file that have not been matched represent the deleted records (step 529). Therefore, the remote segment sends the host segment those unique IDS together with "delete" flags (step 530).

After the remote segment has finished providing data to the host segment, the host segment synchronizes the two databases based on the input from the remote segment. The remote segment waits until the host segment finishes synchronizing and instructs the remote segment in step 409 in Fig. 11 to begin unloading into the remote database (step 532).

The host segment synchronizes the two database similar in the way it synchronizes a so-called "fast synchronization" database (as defined in the '490, '926, and '645 applications) with another database. The operation of a synchronization program synchronizing a fast synchronization database with either a fast synchronization database or a regular database (i.e. non-fast synchronization) is described in detail in the '490, '926, and '645. We will now describe in detail how the information from the remote segment is used to synchronize the remote database with another database.

As described above, a remote segment sending remote database records to the Synchronizer provides field values of only those records which have been changed or added since the previous synchronization but not those records that are unchanged or deleted. Therefore, unlike a regular database Translator, the remote segment does not provide the Synchronizer with unchanged records.

In order to synchronize the remote database with the host database, the Synchronizer transforms information from the remote segment into regarding unchanged records into equivalent regular database records. These transformed records are then used by the Synchronizer in the synchronization. Essentially, the synchronizer transforms and uses the information sent by the remote segment to identify a record in the history file that is a copy of the field values of the unchanged remote database record. In the described embodiment, the synchronizer also copies that history file record and flags as being the remote database record.

The described embodiment uses the host history file to perform this transformation. At the beginning of a first synchronization between the two databases, all records in the remote database are loaded into the host history file. As changes, additions, and deletions are made to the remote database, during each subsequent synchronization, the same changes, additions, and deletions are made to the host history file. Therefore, the host history file at the end of each synchronization will contain a copy of the relevant content of the remote database after synchronization. By relevant, we mean data in the fields that are synchronized. For example, it may be the case that the host history file contain data in fields that are not synchronized. Moreover, if the records of the remote are mapped or recast into another format (e.g. "translated" as described in the '390 patent) the records of the history file contain a copy of the records of the database, as mapped, translated, or both. The Synchronizer uses the mapped or translated records for synchronization. Therefore, it only needs the mapped or translated copy of the unchanged record. In other embodiments, the host history file may contains copies of all the records exactly as they are in

the remote database or in some other format that is useful for the particular application.

Referring to Fig. 13, in the described embodiment, all records received by the host segment from the remote segment are flagged with one of Added, Changed, or Deleted flags. For all records received from the remote segment (step 601), the host synchronizer performs the following functions. If a received record is flagged as an added record (step 602), then the received record is added to the host workspace (step 603). Since the record is new, it is not associated or linked to any history file record. If a record is flagged as a "changed" record (step 604), then the Synchronizer uses the received unique ID to find the corresponding record in the history file (step 605) and links the received remote record to that history file record (step 606). If the received record is flagged as a "deleted" record (step 607), then the Synchronizer uses the received unique ID to find the corresponding record in the history file (step 608)and marks the history file record as deleted (step 609).

After all the received records are analyzed (step 611), if any host history file records containing remote database unique IDs are left that were not matched against the received records, the synchronizer assumes that those records represent the remote database records that are unchanged. For all those records (step 612), the synchronizer clones the host history file record (i.e. create a workspace entry and copy all the host history file record in to that entry) and treats it as a record received from the remote database. At this point the host segment proceeds with synchronization since the records of the remote database have now been loaded. In essence, referring back to Fig. 11, this is the end of step 404.

As previously described, after the synchronizer has performed CAAR, the user must confirm to proceed with updating the remote database (step 406 in Fig. 11). If the user decides to terminate the synchronization, changes are not made to the host history file or the databases. In the case of the remote database, as described in reference to Fig. 12, the remote segment is waiting for the synchronizer to finish synchronizing. If the user aborts synchronization (step 533), the remote segment discards the remote workspace (step 534), saves the original history file without any changes (step 535), and terminates the process at the remote computer.

If the user confirms to proceed with updating the database (step 406 in Fig. 11) , control module 2 instructs the synchronizer and the remote segment to proceed with unloading the records from the workspace into the remote database. As stated, at this point, the remote segment is waiting for the synchronizer to finish synchronizing (step 532 in Fig. 12). During the synchronization, the synchronizer has determined what "actions" with respect to which record in which database should be taken (update, delete, or add) to complete synchronization. If changes or additions are made to the host database in the case of particular record but no action need be taken with respect to that record in the remote database, the synchronizer determines that an "acknowledgement" should be sent to the remote segment. The synchronizer sends all the actions concerning the remote database together with the associated record to the remote (step 616). The synchronizer then sends the unique ID of those records that require "acknowledgements" to be sent to the remote together with an appropriate flag (step 617).

Referring again to Fig. 12, for each action item or acknowledgement received at the remote segment (step

538), the following steps are performed. If the received data indicates an "acknowledgement" or "action" with respect to a record that was added or changed since the previous synchronization, the remote segment marks the new workspace entry that was created in either step 520 or step 525 as acknowledged (step 540). The remote segment also discards or removes any other entry in the workspace that contains the unique ID of this record, which is typically the entry that was loaded from the remote history file. Therefore, as previously described, this entry as opposed to the old remote history file entry associated with this record will be written into the history file at the end of the process at the remote segment. This in essence updates the history file, as will be described below.

If the received data indicates an action item that tells the remote segment to update, change, or add a remote database record (step 543), the remote segment performs that action with respect to the remote database. The remote segment also performs the same steps as steps 540 and 541 (step 544 and 545). If a new record was added to the database (step 546), it will be assigned a new unique ID. The remote segment sends that unique ID to the host segment (step 547). The host segment includes that unique ID in the host work space in association with that record (step 618 in Fig. 13).

After all the records have been received, the remote segment discards all unacknowledged entries from the workspace. Therefore, in the case of those added or changed records with which the user decided not to update the host database, the remote history file remains unchanged. The remote history file is then updated from the remote workspace. At this point the control module continues with step 410 in Fig. 11, i.e. creating the

history file to end the synchronization of the two databases.

In the first embodiment, which we described above, the remote database assigns unique IDs to its records. We will now describe a second embodiment for the case where the remote database does not assign unique IDs to its records. In such a case, the remote segment provides some information less than all the fields of the records to uniquely identify an unchanged record to the host segment. This information may be a hash value. The host segment uses this information to find and then use the host history file copy of the unchanged remote database record to synchronize the two databases.

To identify a record from the previous synchronization or an unchanged record, the remote segment can use a content based code, that is a code whose value depends on the content of all or a selected number of the fields of a record. In the second embodiment, the remote segment uses hash numbers. Since in the case of an unchanged record, its content has remained the same, its hash number remains the same. The hash number acts as a unique identifier and therefore enables the remote and host segments to identify the unchanged record by its hash code. The hash code can be used to identify a record that is stored in the host history file, since its content remains the same from the end of one synchronization to the time it is updated. It may also be transmitted to identify an unchanged record or an unchanged version of a changed record. A host history file record can in effect be identified using the hash code of that record.

We will describe the operation of this embodiment in reference to Figs. 14 and 15. Steps 701 -711 are the same as steps 501-511 in Fig. 12, described above in reference to the first embodiment. These steps are

generally concerned with finding the correct remote history file.

After determining that there is a suitable remote history file, for each record of the remote database (step 712), the following functions are performed. The remote segment loads and translates a record of the remote database into the remote workspace (step 713) and a hash number is calculated for that record (step 714). If the hash number of the remote record matches one or more hash numbers in the remote history file (step 715), then the remote segment assumes that the record has not been changed since a previous synchronization.

It is possible that the hash number may be repeated more than once, e.g. because of duplicate records or records that appear as duplicates because some of their fields are not synchronized. Therefore, the remote segment sends additional information that can be used to identify which of the multiple identical hash numbers a particular record relates to. This is done because during updating the remote history file record at the end of synchronization, the same number of identical hash numbers as matching remote database records are updated. In the second embodiment, this additional information is the index number associated with each entry of the remote workspace. Therefore, when the hash number of the remote record matches one or more hash numbers in the remote history file (step 715), the remote segment sends the hash number, a flag indicating that the record is unchanged, and the index number of that hash number to the host segment (step 716). Obviously if the index number was previously sent, the next index number for the identical hash is sent.

If the hash number does not match one or more hash numbers in the history file (step 717), the remote segment treats that record as having been newly added.

Therefore, the remote segment sends the host segment a copy of the field values of the record, the remote workspace index number, and an "added" flag (step 720). The remote workspace index number makes it easier to perform future search of the remote workspace when data with respect to this record is received. As in the case of changed and added record in the first embodiment, the remote segment also creates a new remote workspace entry and enters hash number value of the record (step 718). The new entry is marked as "unacknowledged" (step 719). It should be noted that although the remote segment treats the record as a new record, the remote segment can not distinguish between an added and a changed record. Therefore, the synchronizer during synchronization does not treat it as a new record. Instead, the synchronizer compares the record to determine whether it matches with any of host history file record which would mean it is a changed record.

After reading all the remote database records and processing them (step 722), the remote segment removes from the remote workspace all entries that have hash numbers that are unmatched (step 723). These entries represent records that have either been changed or deleted since the previous synchronization.

After the remote segment has finished providing data to the host segment, the host segment synchronizes the two databases based on the input from the remote segment. The remote segment waits until the host segment finishes synchronizing and instructs the remote segment in step 409 in Fig. 11 to begin unloading into the remote database (step 724).

Referring to Fig. 15, as in the case of the first embodiment, the synchronizer on the host computer uses the information to identify those records in the host history file that correspond to the unchanged remote

database records. For every record received from the remote segment that is flagged as added (step 801), the synchronizer adds the record to the host workspace (step 802) and during CAAR compares the record to the history file to determine whether the record is a changed or added record. For every record received from the remote segment that is flagged as "unchanged" (step 804), in the same manner as the first embodiment, the synchronizer finds the corresponding host history file record by finding a record that has the same hash number as that sent by the remote synchronizer (step 805). The synchronizer then clones the record (step 806), as previously described, and treats as if it is a record received from the remote database. At the end of this process, when all the records of the remote database are loaded into the host workspace, the control module proceeds to step 405 in Fig. 11 to begin CAAR. CAAR will then analyze the records in the host workspace to determine which remote records were added, which were changed, and which were deleted since the previous synchronization.

After CAAR, if the user confirms to proceed with updating the database, control module 2 instructs the synchronizer and the remote segment to proceed with unloading the records from the workspace into the remote database (step 409 in Fig. 11). As stated, at this point, the remote segment is waiting for the synchronizer to finish synchronizing (step 724 in Fig. 14). During performing CAAR, the synchronizer has determined what actions should be taken (update, delete, or add) to each database. If changes or additions are made to the host database in the case of a particular record but no action need be taken with respect to that record in the remote database, the synchronizer determines that at least an "acknowledgement" is to be sent to the remote segment.

The synchronizer sends all the actions concerning the remote database together with the associated record and remote workspace index to the remote (step 809). The synchronizer then sends the remote workspace index of those records that require acknowledgements to be sent to the remote together with an appropriate flag (step 810). Therefore, the remote workspace index is used to identify which records in the remote workspace should be "acknowledged".

Referring back to Fig. 14, steps 725-729 are the same as steps 533-537, which were described in reference to the first embodiment. For each action item or acknowledgement received at the remote segment (step 730), the following steps are performed. If the data received indicates an "acknowledgement" or "action" with respect to a record that was sent to the host segment flagged as "added" (step 731), the remote segment marks the new workspace entry that was created in either step 718 as acknowledged (step 732). It should be noted that the remote workspace index number is used to locate the remote workspace entry. Therefore, as previously described, this entry will be written into the history file at the end of the process at the remote segment.

If the received data indicates an action item that tells the remote segment to update, change, or add a remote database record (step 733), the remote segment performs that action with respect to the remote database. The remote segment also updates the remote workspace and marks the entry as "acknowledge" (step 735).

After all the records have been received, the remote segment discards all unacknowledged entries from the workspace, which were newly created entries which were not acknowledged. Therefore, in case of those added or changed records with the user decided not to update the host database with, the remote history file remains

unchanged. The remote history file is then updated from the workspace. At this point the control module continues with step 410 in Fig. 11, i.e. creating the history file to end the synchronization of the two databases.

Although we have described embodiments in which the host segment transforms the input from the remote segment, it should be noted that other embodiments of the host segment may not transform the input from the remote segment since they are designed to use inputs that informs them of which records have been changed, added and deleted or have been left unchanged. Other embodiments in which the host segment requires different types of input, the input from the remote segment are transformed as required. The various embodiments of the host segment may or may not use a history file.

Other embodiments are within the following claims.

What is claimed is:

1. A computer implemented method of synchronizing a first and a second database comprising:

determining whether the records of the first database have been changed or added since the previous synchronization, based on data reflecting whether the records of the first database have been added or changed since a previous synchronization;

if one of the records of the first database has not been changed or added since the previous synchronization, performing a synchronization with records of the second database using a record representative of the one record at the time of a previous synchronization, the representative record being stored in a history file containing records reflecting the contents of records of the databases at the time of a previous synchronization.

2. The computer implemented method of claim 1, wherein the data comprises database generated data for keeping track of whether the records of the first database have been added or changed since a previous synchronization.

3. The computer implemented method of claim 2 wherein the computer generated data indicates the most recent date and time of when the records were created or changed.

4. The computer implemented method of claim 2 wherein the computer generated data comprises a flag set when the records are created or changed.

5. The computer implemented method of claim 2 wherein the first database provides further database generated data indicating which of the records were deleted since the previous synchronization, further comprising:

identifying records of the history file corresponding to the deleted records by performing a comparison of the further computer generated data with the history file; and

completing the synchronization using a result of the identification.

6. The computer implemented method of claim 2 further comprising:

identifying records of the history file corresponding to records of the first database deleted since the previous synchronization by performing a comparison of the records of the history file and a result of determining whether the records of the first database have been added or changed.

7. The computer implemented method of claim 2 wherein the first database assigns a unique identification data to the records of the first database, the method further comprising:

performing a comparison of the records of the history file and the unique identification data of records in the first database that have been changed or added since the previous synchronization; and

completing synchronization using a result of the comparison.

8. The computer implemented method of claim 2 further comprising:

modifying, adding, or deleting the records of the first database based on a result of the synchronization.

9. The computer implemented method of claim 2 further comprising modifying, adding, or deleting records in the history file using results of the synchronization such that the history file contains records reflecting the contents of records of the databases after the synchronization.

10. A computer program, resident on a computer readable medium, for synchronizing a first and a second database, comprising instructions for:

determining whether the records of the first database have been changed or added since the previous synchronization, based on data reflecting whether the records of the first database have been added or changed since a previous synchronization;

if one of the records of the first database has not been changed or added since the previous synchronization, performing a synchronization with records of the second database using a record representative of the one record at the time of a previous synchronization, the representative record being stored in a history file containing records reflecting the contents of records of the databases at the time of a previous synchronization.

11. The computer program of claim 10, wherein the data comprises database generated data for keeping track of whether the records of the first database have been added or changed since a previous synchronization.

12. The computer program of claim 11 wherein the computer generated data indicates the most recent date and time of when the records were created or changed.

13. The computer program of claim 11 wherein the computer generated data comprises a flag set when the recorda are created or changed.

14. The computer program of claim 11 wherein the first database provides further database generated data indicating which of the records were deleted since the previous synchronization, further comprising instructions for:

identifying records of the history file corresponding to the deleted records by performing a comparison of the further computer generated data with the history file; and

completing the synchronization using a result of the identification.

15. The computer program of claim 11 further comprising instructions for:

identifying records of the history file corresponding to records of the first database deleted since the previous synchronization by performing a comparison of the records of the history file and a result of determining whether the records of the first database have been added or changed.

16. The computer program of claim 11 wherein the first database assigns a unique identification data to the records of the first database, the computer program further comprising instructions for:

performing a comparison of the records of the history file and the unique identification data of records in the first database that have been changed or added since the previous synchronization; and

completing synchronization using a result of the comparison.

17. The computer program of claim 11 further comprising instructions for:

modifying, adding, or deleting records of the first database based on a result of the synchronization.

18. The computer program of claim 11 further comprising instructions for modifying, adding, or deleting records in the history file using results of the synchronization such that the history file contains records reflecting the contents of records of the databases after the synchronization.

19. A computer implemented method for synchronizing a first database located on a first computer and a second database located on a second computer, the method comprising:

determining whether a record of the first database has been changed or added since a previous synchronization;

if the record of the first database has not been changed or added since the previous synchronization, sending from the first computer to the second computer information which the second computer uses to identify the record of the first database to be unchanged, wherein a history file located on the second computer contains records reflecting the content of records of the first database at the completion of a previous synchronization and said information identifies a record in the history file reflecting the content of the record of the first database; and

synchronizing the first database and the second database using the sent information.

20. A computer program, resident on a computer readable medium for synchronizing a first database located on a first computer and a second database located on a second computer, comprising instructions for:

determining whether a record of the first database has been changed or added since a previous synchronization;

if the record of the first database has not been changed or added since the previous synchronization, sending from the first computer to the second computer information which the second computer uses to identify the record of the first database to be unchanged, wherein a history file located on the second computer contains records reflecting the content of records of the first database at the completion of a previous synchronization and said information identifies a record in the history file reflecting the content of the record of the first database; and

synchronizing the first database and the second database using the sent information.

21. A computer implemented method for synchronizing a first database located on a first computer and a second database located on a second computer, the method comprising:

determining, at the first computer, whether a record of the first database has been changed or added since a previous synchronization, using a first history file located on the first computer comprising records representative of records of the first database at the completion of the previous synchronization;

if the record of the first database has not been changed or added since the previous synchronization, sending from the first computer to the second computer information which the second computer uses to identify the record of the first database to be unchanged.

22. The computer implemented method of claim 21 wherein a second history file located on the second computer contains records representative of records of the first database at the completion of the previous synchronization, wherein one of the representative records represents the record of the first database determined to be unchanged, and the method further comprises performing a synchronization, at the second computer, of the second and first databases using the one of the representative records.

23. The computer implemented method of claim 22 wherein the information sent from the first computer to the second computer is used to locate the one of the representative records in the second history file.

24. The computer implemented method of claim 23 wherein the second history file stores information in relation to the representative records and wherein the one of the representative records in the second history file can be identified from the stored information.

25. The computer implemented method of claim 24 wherein the information sent from the first computer to the second computer comprises information that matches the information stored in relation to the one of the representative records in the second history files.

26. The computer implemented method of claim 21 wherein the information comprises information identifying records other than the unchanged record.

27. The computer implemented method of claim 21 wherein the information comprises information identifying the unchanged record.

28. The computer implemented method of claim 21 wherein the information comprises information identifying the deleted records.

29. The computer implemented method of claim 21 wherein the information comprise information identifying the added records.

30. The computer implemented method of claim 21 wherein the information comprises a code, the code being based on at least a portion of the content of the record of the first database.

31. The computer implemented method of claim 30 wherein the code comprises a hash number computed based on at least a portion of the content of the record of the first database.

32. The computer implemented method of claim 30 wherein the information further comprises a first plurality of records of the first database identified as "changed or added", the method further comprises using said information to indentify a plurality of the first database as "deleted or changed" since the previous synchronization.

33. The computer implemented method of claim 21 wherein the information comprises a code uniquely identifying the records of the first database.

34. The computer implemented method of claim 33 wherein the unique identification code is assigned by the first database to the records of the first database.

35. The computer implemented method of claim 34 wherein the information further comprising a first plurality of the records of the first database identified as "changed", a second plurality of the records of the first database identified as added, and information identifying a third plurality of records of the first database as "deleted".

36. A computer implemented method of identifying a record of a database stored on a first computer to a second computer comprising:

reading a record of the database;

assigning a code to the record of the database, the code being based on at least a portion of the content of the record of the first database;

transmitting the code to the second computer to identify the record to the second computer.

37. The computer implemented method of claim 36 wherein the code comprises a hash number computed based on at least a portion of the content of the record of the first database.

38. A computer program, resident on a computer readable medium for synchronizing a first database located on a first computer and a second database located on a second computer, comprising instructions for:

determining, at the first computer, whether a record of the first database has been changed or added since a previous synchronization, using a first history file located on the first computer comprising records representative of records of the first database at the completion of the previous synchronization;

if the record of the first database has not been changed or added since the previous synchronization, sending from the first computer to the second computer information which the second computer uses to identify the record of the first database to be unchanged.

39. The computer program of claim 38 wherein a second history file located on the second computer contains records representative of records of the first database at the completion of the previous synchronization, wherein one of the representative records represents the record of the first database determined to be unchanged, and the program further comprising instructions for performing a synchronization, at the second computer, of the second and first databases using the one of the representative records.

40. The computer program of claim 39 wherein the information sent from the first computer to the second computer is used to locate the one of the representative records in the second history file.

41. The computer program of claim 40 wherein the second history file stores information in relation to the representative records and wherein the one of the representative records in the second history file can be identified from the stored information.

42. The computer program of claim 41 wherein the information sent from the first computer to the second computer comprises information that matches the information stored in relation to the one of the representative records in the second history files.

43. The computer program of claim 38 wherein the information comprises information identifying records other than the unchanged record.

44. The computer program of claim 38 wherein the information comprises information identifying the unchanged records.

45. The computer program of claim 38 wherein the information comprises information identifying the deleted records.

46. The computer program of claim 38 wherein the information comprise information identifying the added records.

47. The computer program of claim 38 wherein the information comprises a code, the code being based on at least a portion of the content of the record of the first database.

48. The computer program of claim 47 wherein the code comprises a hash number computed based on at least a portion of the content of the record of the first database.

49. The computer program of claim 47 wherein the information further comprises a first plurality of records of the first database identified as "changed or added", the program further comprising instructions for using said information to indentify a plurality of the first database as "deleted or changed" since a previous synchronization.

50. The computer program of claim 38 wherein the information comprises a code uniquely identifying the record of the first database.

51. The computer program of claim 50 wherein the unique identification code is assigned by the first database to the record of the first database.

52. The computer program of claim 50 wherein the information further comprises a first plurality of the records of the first database identified as "changed", a second plurality of the records of the first database identified as added, and information identifying a third plurality of records of the first database as "deleted".

53. A computer program, resident on a computer readable medium, for identifying a record of a database stored on a first computer to a second computer comprising instructions for:

reading a record of the database;

assigning a code to the record of the database, the code being based on at least a portion of the content of the record of the first database;

transmitting the code to the second computer to identify the record to the second computer.

54. The computer program of claim 53 wherein the code comprises a hash number computed based on at least a portion of the content of the record of the first database.

55. A computer implemented method of synchronizing at least a first and a second database, wherein the first database provides information regarding records which have been added, deleted or modified since a previous synchronization, the method comprising:

storing in a history file records representative of the records of the first database after the previous synchronization;

obtaining information regarding the records which have been modified, deleted, or added since the previous synchronization;

performing a first comparison of the records of the second database with records in history file corresponding to records of the first database that have not been modified, added, or deleted;

completing the current synchronization based on the outcome of the first comparison and the information; and

modifying, adding, or deleting records in the history file using the information obtained from the first database such that the history file contains records representative of the records of the first database after the current synchronization.

56. A computer program, resident on a computer readable medium, for synchronizing at least a first and a second database, wherein the first database provides information regarding records which have been added, deleted or modified since a previous synchronization, comprising instructions for:

storing in a history file records representative of the records of the first database after the previous synchronization;

obtaining information regarding the records which have been modified, deleted, or added since the previous synchronization;

performing a first comparison of the records of the second database with records in history file corresponding to records of the first database that have not been modified, added, or deleted;

completing the current synchronization based on the outcome of the first comparison and the information; and

modifying, adding, or deleting records in the history file using the information obtained from the first database such that the history file contains records representative of the records of the first database after the current synchronization.

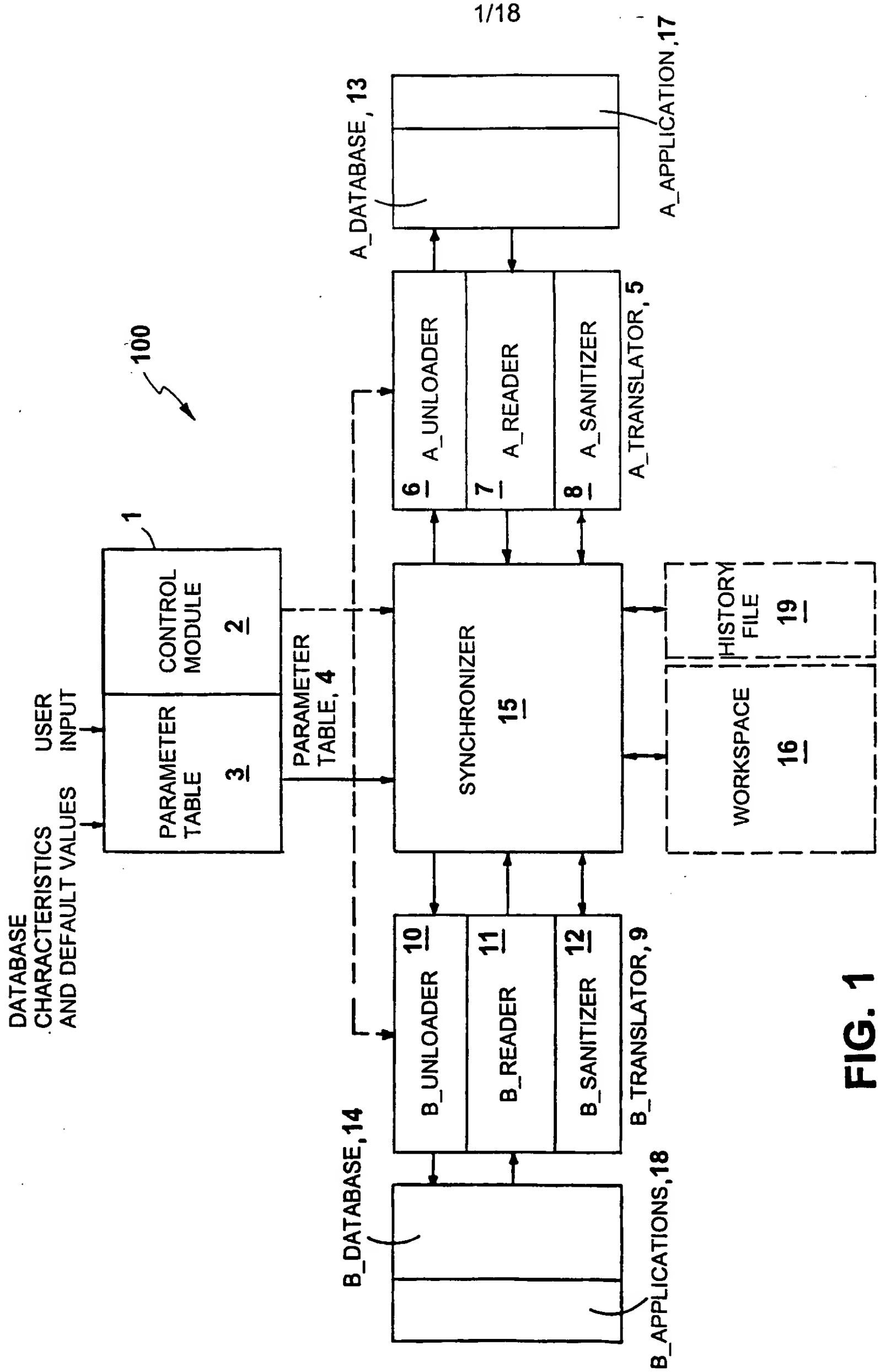


FIG. 1

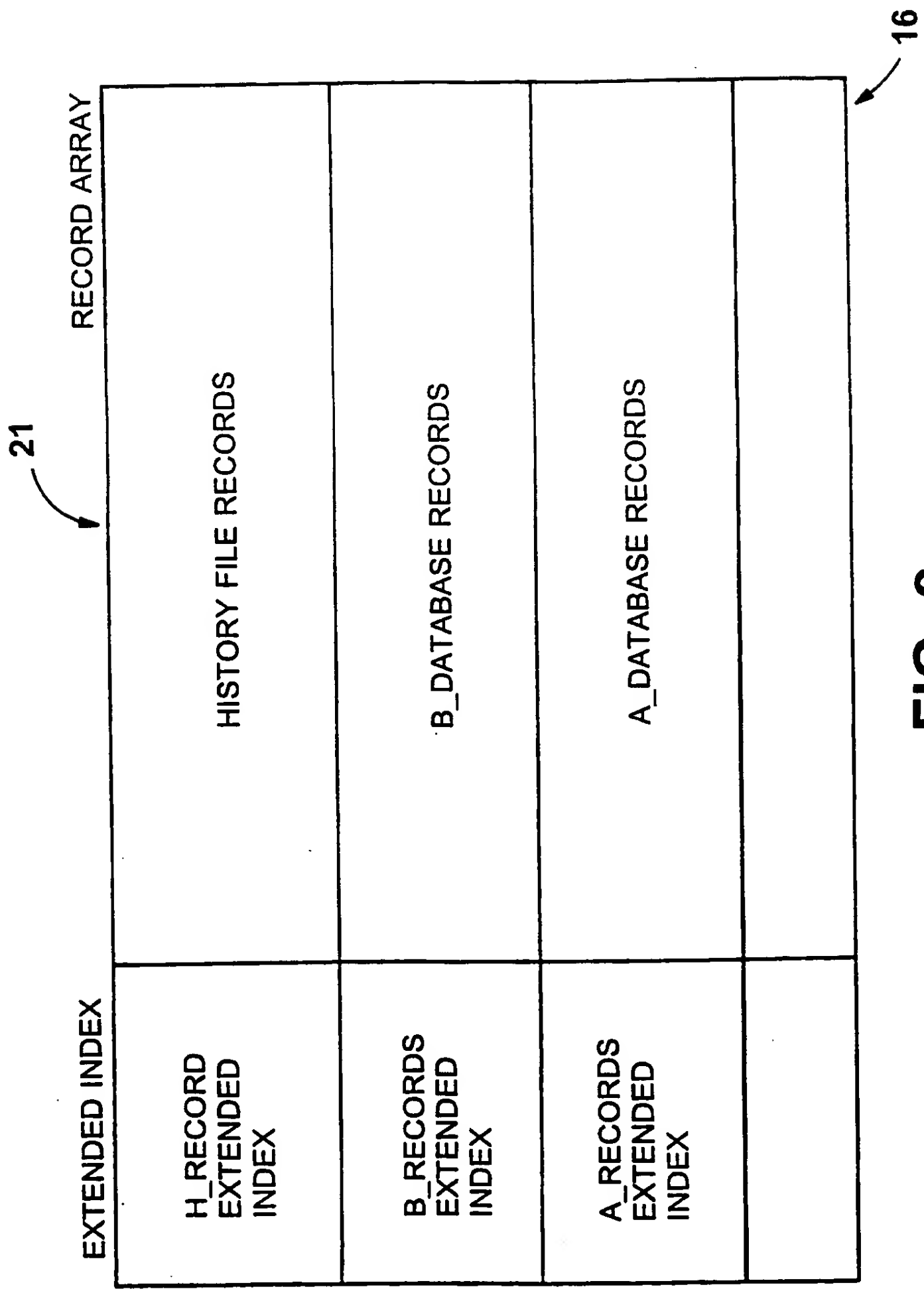


FIG. 2

Pseudo Code for Translation Engine Control Module

100. CREATE Parameter_Table from User Input A & B database characteristics and default values
101. INSTRUCT Synchronizer to initialize itself
102. INSTRUCT Synchronizer to LOAD the History_File into its WORKSPACE
103. INSTRUCT B_Translator to LOAD all of B_records from B_Database and SEND to Synchronizer (Synchronizer STORES these records in WORKSPACE)
104. INSTRUCT A_Translator to SANITIZE B_records that were just LOADED (A_Translator USES Synchronizer services to read and write records in the WORKSPACE; Synchronizer maps these records using the B-A_Map before sending them to A_Translator and maps them back using A-B_Map before rewritting them into the WORKSPACE)
105. INSTRUCT A_Translator to LOAD all of A_records form A_Database and SEND to Synchronizer (Synchronizes STORES these records in WORKSPACE by first mapping them using the A-B_Map and them storing in their new form)
106. INSTRUCT B_Translator to SANITIZE A_records that were just LOADED (B_Translator uses Synchronizer services to read and write records in the WORKSPACE)
107. INSTRUCT Synchronizer to do CAAR (Conflict Analysis And Resolution) on all the records in WORKSPACE.
108. INFORM user exactly what steps Synchronizer proposes to take (i.e. Adding, Changing, and Deleting records). WAIT for User
109. IF user inputs NO, THEN ABORT
110. INSTRUCT B_Translator to UNLOAD all applicable records to B_Database.
111. INSTRUCT A_Translator to UNLOAD all applicable records to the A_Database.
112. INSTRUCT Synchronizer to CREATE a new History File.

FIG. 3

1050. Verify History File
1051. If verified, Then Proceed as Fast Synch
1052. If not, Then Proceed as Synchronization from Scratch load all record in databasse

1053. If Fast Synch
1054. LOAD records into the Workspace. Map if necessary
1055. Sanitize Records not marked as Deletion
1056. Orientation analysis (Fig. 11).
1057. For each H_Record, analyze the CIG that the H_Record belongs to.
1058. IF the H_Record's CIG contains no Record from the Fast Synchronization database, THEN CLONE the H-Item, label it a Fast Synchronization Record, and add it to the H_Record's CIG.
1059. If the H_Record's CIG contains a Fast Synchronization record that is marked as a a Deletion, it is now removed from the CIG.
1060. If the H_Record's CIG contains a non-Delete Fast Synchronization Record, then do nothing.
1061. END LOOP

FIG. 4

1150. Verify History File
1151. If verified, Then Proceed as Fast Synch
1152. If not, Then Proceed as Synchronization from Scratch

1153. IF synchronization from scratch
1154. IF record outside of current_date_range THEN MARK record as out-of-range

1155. If Fast Synch
1156. Load History File into Workspace
1157. MARK History File records outside of previous_date_range as Bystander
1158. Load All Fast Synchronization Records into the Workspace; mapped if necessary.
1159. SANITIZE Records which are not DELETEs
1160. Orientation analysis (Fig. 11).
1161. If Added Fast Synchrnization record is out of current date range THEN MARK Out-Of_Range
1162. If Changed or deleted Fast Synchronization record in a CIG with Bystander H_Record, MARK the Bystander record as Garbage
1163. For each H_Record, analyze the CIG that the H_Record belongs to.
1164. If the H_Record's CIG contains no Record from the Fast Synchronization database, then make a clone of the H-Item, label it a Fast Synchronization Record, and adding it to the H_Record's CIG.
1165. If H_Record is not a Bystander, THEN Make a clone of H_Record, mark as Fast

FIG. 5A

PCT/US97/20627

```
                Synchronization record, and Add to CIG
1166.           IF H_Record is Bystander THEN
1167.                   IF outside of Current date range THEN Do Nothing
1168.                   ELSE {Within Current Date Range}
1169.                           Mark H_Record as Garbage, Clone H_Record and Mark it as from
                                Fast Synchronization database
1170.                   END IF
1171.           END IF
1172.           If the H_Record's CIG contains a Fast Synchronization record that is marked as a
                deletion, it is now removed from the CIG.
1173.           If the H_Record's CIG contains a non-deletion Fast Synchronization Record, then do
                nothing.
1174.           Any Fast Synchronization records which are not joined to any H_Record's CIG
                represent additions and no transformation is required.
1175.   END LOOP
```

FIG. 5B

```
1200    FOR each Record in Database
1201        Get Unique ID
1202        Get Last modified Date/Time
1203            IF Last modified Date/Time Stamp (Less Than or Equal To) Last Sync Date/Time Stamp
1204                Set Flag to indicate "Unchanged" Record
1205            ELSE
1206                Set Flag to indicate "Changed/New" Record
1207                FOR each Field in Record
1208                    Load Field Data into the Workspace
1209                NEXT
1210            ENDIF
1211        Return record to the Sync Engine [Please explain.what this line does]
1212        NEXT

        In the synchronizer:
1213     Compare the loaded unique IDs with the Unique ID of all records in the history file
1214    Determine as deleted those records whose unique ID of records in the history file but not in loaded records
1215    Compare unique IDs of records marked as changed to unique IDs of the records of the history file.
1216    Determine as Added those records having unique IDs not present in the history file
1217    Proceed as in fast synchronization to synchronize
1218    Update the history file as for fast synchronization databases
1219    Unload records as for fast synchronization databases
```

FIG. 6

```
1250    Query the Database for All Records where Last Modified date/time <= Last Sync Date/Time
1251    FOR each Record in QuerySet
1252            Get Unique ID
1253            Set Flag to indicate "Unchanged" Record
1254            Return record to the Sync Engine
1255    NEXT
1256    Query the Database for All Records where Last Modified date/time > Last Sync Date/Time
1257    FOR each Record in QuerySet
1258            Get Unique ID
1259            Set Flag to indicate "Changed/New" Record
1260            FOR each Field in Record
1261                    Load Field Data into the Workspace
1262            NEXT
1263            Return record to the Sync Engine
1264    NEXT
```

FIG. 7

```
1350.  For each Record in Database
1351.          Get unique ID
1352.          Get Last modified Date/Time or Dirty bit, as applicable
1353.          If the record has not been changed since the previous synchronization then
1354.                  Call Synchronizer's PutUnchangedRecord function with unique ID
1355.                  In the Synchronizer: in reponse to the PutUnchangedRecord fucntion call, use unique
                       ID to find matching History file record using unique ID
1356.                  Clone History record
1357.          Else
1358.                  For each Field in Record
1359.                          Load Field Data
1360.                  Next Field
1361.          Put loaded record in the workspace.
1362.  Next Record.

       In the Synchronizer:
1363.  Synchronize the two database as if the records were loaded from a regular (i.e. non-fast synchronization
       database)
```

FIG. 8

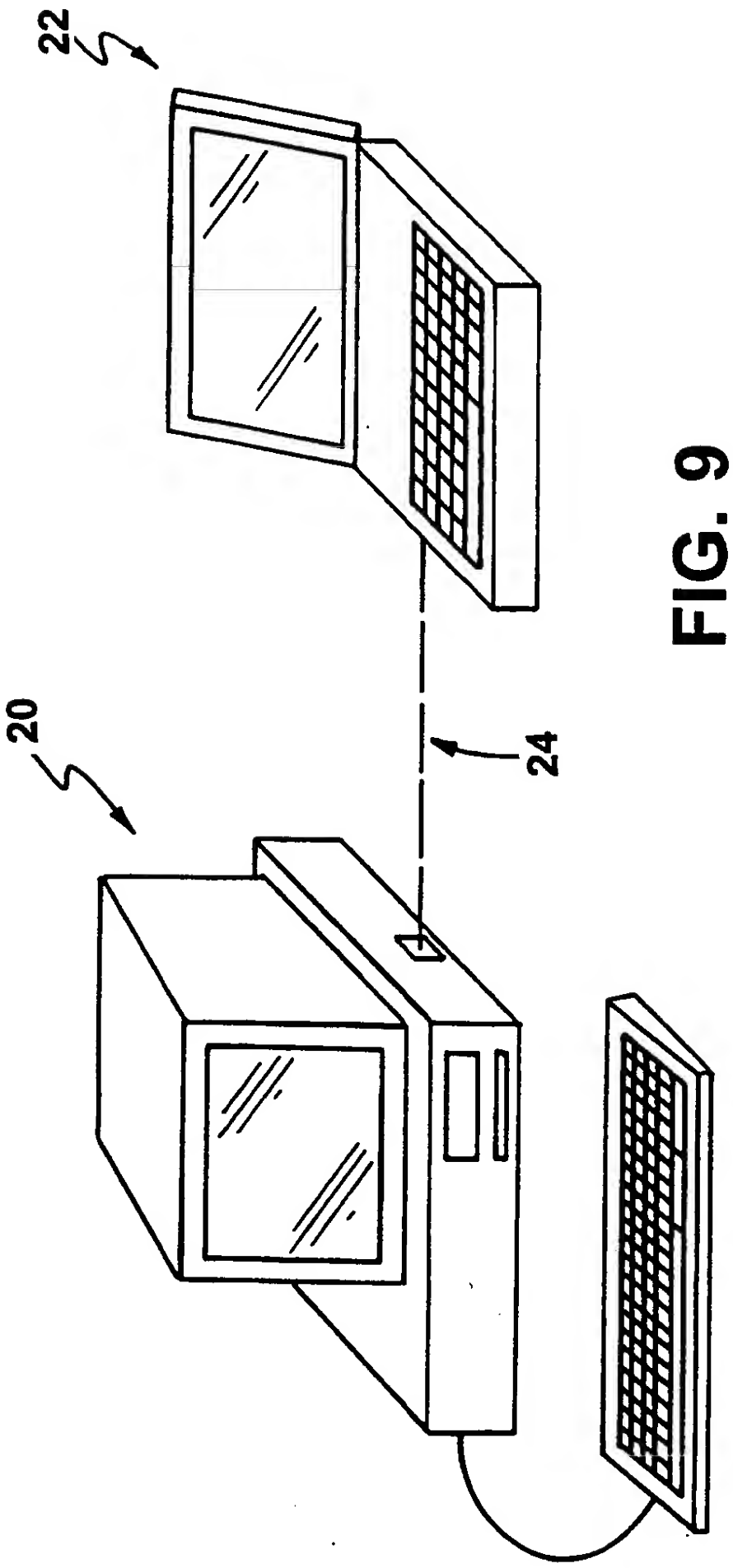


FIG. 9

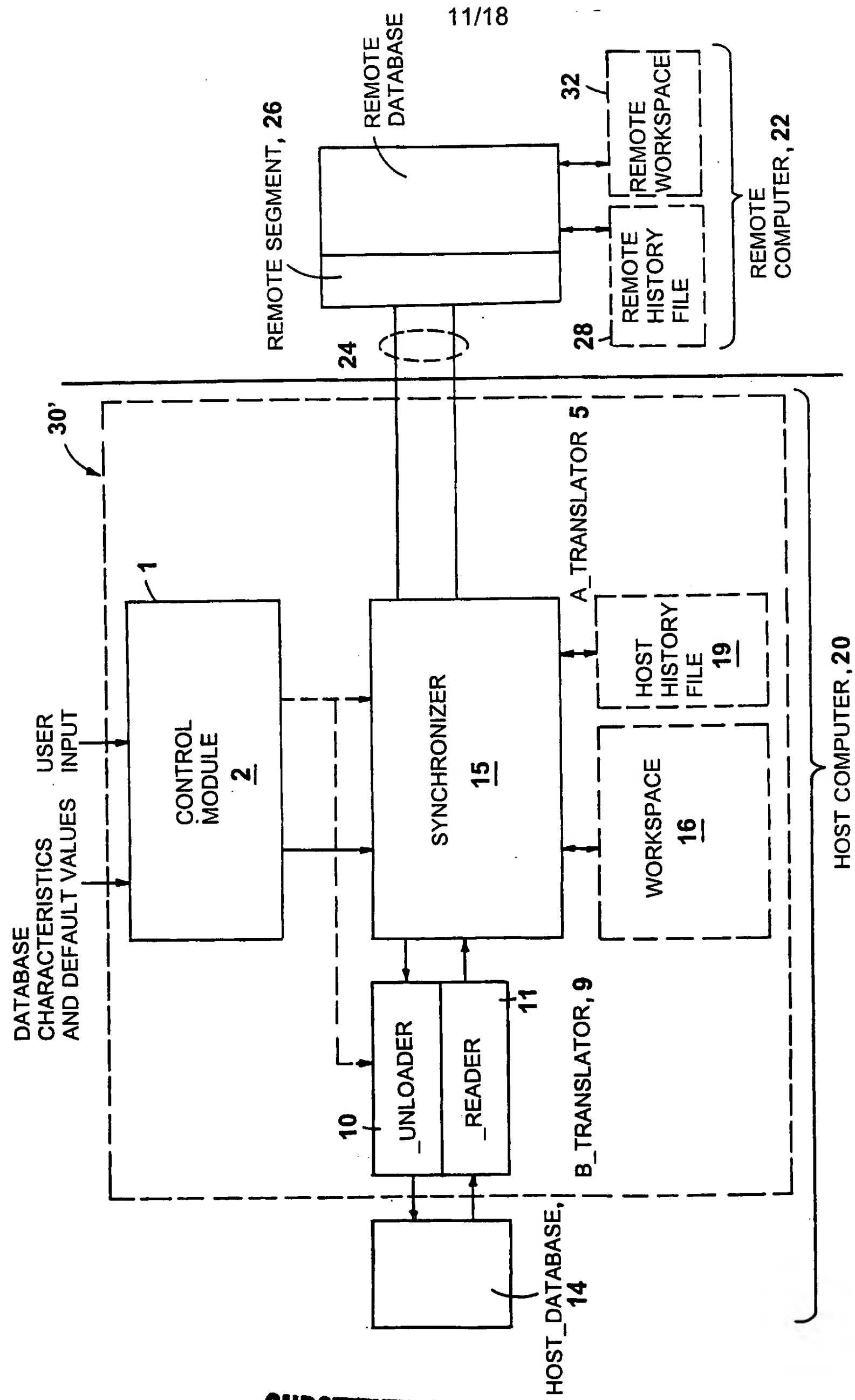


FIG. 10

401. INSTRUCT the synchronizer to initialize itself, specifying current user options
402. INSTRUCT the synchronizer to load the appropriate history file into its workspace
403. INSTRUCT host Translator to LOAD all of host Records from host Database
404. INSTRUCT RemoteTranslator to LOAD all of Remote Records from Remote Database --
405. INSTRUCT Synchronizer to perform (CAAR) on all the history, B records, and A records in the workspace.
406. INFORM user exactly what changes are about to be made
407. If user inputs NO, THEN ABORT
408. INSTRUCT host Translator to UNLOAD all applicable changes to host Database.
409. INSTRUCT Remote Translator to UNLOAD all applicable changes to Remote Database.
410. INSTRUCT Synchronizer to CREATE a new history file.

FIG. 11

```
501.  INITIALIZE an empty remote workspace
502.  IF there is a remote history file matching the host history file name
503.        IF the remote history file time stamp matches the history file time stamp
504.              LOAD the remote history file into the remote workspace
505.        ELSE
506.              REMOVE the non-matching history file
507.              Proceed with the empty workspace, all records passed to host
508.        ENDIF
509.  ELSE
510.        Proceed with the empty workspace, all records passed to host
511.  ENDIF
512.  FOR each record in the remote database
513.        Translate and load data field values and unique ID into remote workspace
514.        Compute a hash value to represent all translated data values
515.        IF the unique ID matches the unique ID of an existing remote history file entry,
516.              IF the hash value is the same
517.                    Skip this entry, the host will recreate this record from history
518.              ELSE
519.                    Send Unique ID, field values and "Changed" record flag to the host
520.                    Create new workspace entry with same unique ID and new hash value
521.                    This new entry is marked as "unacknowledged"
522.              ENDIF
```

FIG. 12A

```
523.            ELSE
524.                    Send Unique ID, field values and "Added" record flag to the host
525.                    Create new workspace entry with new unique ID and new hash value
526.                    This new entry is marked as "unacknowledged"
527.            ENDIF
528.    NEXT
529.    FOR each unique ID in the remote history file not matched in the above loop,
530.            Send Unique ID and "Deleted" flag to the host
531.    NEXT
532.    WAIT for host to synchronize the data and for user to confirm results
533.    IF user has aborted the synchronization
534.            The remote workspace is discarded.
535.            The original remote history file remains unmodified.
536.            The process is terminated.
537.    ENDIF
538.    FOR each record "action" or "acknowledgment" received from the host,
539.            IF this is an acknowledgment of a record Added or Updated in the remote database,
540.                    Mark any corresponding, newly created workspace item as "acknowledged"
541.                    Remove any prior workspace item with the same unique ID
542.            ELSE IF this is a new action to Add, Update, or Delete a remote database record
543.                    UPDATE remote workspace to reflect the appropriate change
544.                    Mark any corresponding, newly created workspace item as "acknowledged"
545.                    Remove any prior workspace item with the same unique ID
546.                    IF this is an Add
547.                            SEND the new unique ID back to the host to include in history file
548.                    ENDIF
549.            ENDIF
550.    NEXT
551.    REMOVE any newly create, but "unacknowledged" entries from the workspace
552.    UPDATE the remote history file from the remote workspace
```

FIG. 12B

Recreation of "filtered" data by the synchronizer using history data when unique ID's are present.

```
601.    FOR all Added, Changed, and Deleted records transmitted from the remote .
602.            IF record was flagged "Added"
603.                    Add the new record to the workspace, not correlated with any history at this time
604.            ELSE IF record has been changed
605.                    Find corresponding history item in the workspace by the changed record's Unique ID
606.                    Associate (link) the changed record with this workspace item
607.            ELSE IF record has been deleted
608.                    Find corresponding history item in the workspace by the deleted record's Unique ID
609.                    Associate (link) the delete action with this workspace item
610.            ENDIF
611.    NEXT
612.    FOR all history records not yet matched by a remote Unique ID,
613.            CLONE the missing record data and Unique ID from data in the history file
614.    NEXT
```

Updating the remote computer's remote history file while unloading changes from the workspace to the remote database (when unique ID's are used):

```
615.    FOR all actions or acknowledgments recorded in the workspace for the remote database
616.            SEND all actions with associated record data to the remote
617.            SEND all acknowledgments to the remote with associated unique ID's
618.            UPDATE workspace with unique ID's sent from the remote as the result "Add" actions.
619.    NEXT
```

FIG. 13

```
701.  INITIALIZE an empty remote workspace
702.  IF there is a remote history file matching the host history file name
703.        IF the remote history file time stamp matches the history file time stamp
704.              LOAD the remote history file into the remote workspace
705.        ELSE
706.              REMOVE the non-matching history file
707.              Proceed with the empty workspace, all records passed to host
708.        ENDIF
709.  ELSE
710.        Proceed with the empty workspace, all records passed to host
711.  ENDIF
712.  FOR each record in the remote database,
713.        Translate and load data field values into the remote workspace
714.        Compute a hash value to represent all translated data values
715.        IF the hash value matches the hash value of one or more remote history file entries,
716.              Send hash value, remote workspace index, and "Unchanged" record flag to the host
717.        ELSE
718.              Create new workspace entry with new new hash value and remote workspace index
719.              This new entry is marked as "unacknowledged"
720.              Send hash value, remote workspace index, field values and "Added" record flag to the
```

FIG. 14A

```
host
721.            ENDIF
722.    NEXT
723.    REMOVE any "prior" workspace entries not matched by hash value above
724.    WAIT for host to synchronize the data and for user to confirm results
725.    IF user has aborted the synchronization
726.            The remote workspace is discarded.
727.            The original remote history file remains unmodified.
728.            The process is terminated.
729.    ENDIF
730.    FOR each record "action" or "acknowledgment" received from the host,
731.            IF this is an acknowledgment of a record sent to the host (above) as "added",
732.                    Mark any corresponding, newly created workspace item as "acknowledged"
733.            ELSE IF this is a new action to Add, Update, or Delete a remote database record
734.                    UPDATE remote workspace to reflect the appropriate change
735.                    Mark the updated record as "acknowledged"
736.            ENDIF
737.    NEXT
738.    REMOVE any newly create, but "unacknowledged" entries from the workspace
    UPDATE the remote history file from the remote workspace
```

FIG. 14B

Recreation of "filtered" data by the synchronizer using history data when unique ID's are not present.

```
801.    FOR all Added records transmitted from the remote ,
802.            Add the new record to the workspace, not correlated with any history at this time
803.    NEXT
804.    FOR all Unchanged items transmitted from the remote ,
805.            Find corresponding history item in the workspace by the unchanged record's hash value
806.            CLONE the missing record data from data in the history file
807.    NEXT
```

Updating the remote computer's remote history file while unloading changes from the workspace to the remote database (when unique ID's are not used):

```
808.    FOR all actions or acknowledgments recorded in the workspace for the remote database
809.            SEND all actions with associated record data to the remote
810.            SEND all acknowledgments to the remote with associated remote workspace indexes
811.    NEXT
```

FIG. 15